EFFICIENT AND SECURE DOCUMENT SIMILARITY SEARCH OVER CLOUD UTILIZING MAPREDUCE

by Mahmoud Alewiwi

Submitted to the Graduate School of Engineering and
Natural Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Sabanci University
December, 2015

EFFICIENT AND SECURE DOCUMENT SIMILARITY SEARCH OVER CLOUD UTILIZING MAPREDUCE

APPROVED BY:

Prof.Dr. Erkay SAVAŞ .....
(Thesis Supervisor)

Prof.Dr. Yücel SAYGIN .....
(Internal Examiner)

Assoc.Prof.Dr. Kemal KILIÇ .....
(Internal Examiner)

Asst.Prof.Dr. Selçuk BAKTIR .....
(External Examiner)

Asst.Prof.Dr. Ahmet Onur DURAHIM .....
(External Examiner)

DATE OF APPROVAL: .....

Acknowledgments

I wish to express my gratitude to my supervisor Erkay Savaş for his invaluable guidance, support and patience all through my thesis. I am also grateful to Cengiz Örencik for his guidance and valuable contributions to this thesis.

Special thanks to my colleague Ayşe Selçuk, for her collaborating in administrating the Hadoop framework and her kind suggestions.

I am grateful to all my friends from Cryptography and Information Security Lab. (i.e., FENS 2001), Sabanci University and Data Security and Privacy Lab for being very supportive.

I am indebted to the members of the committee of my thesis for reviewing my thesis and providing very useful feedback.

I am grateful to TÜBİTAK (The Scientific and Technological Research Council of Turkey), for the support under Grant Number 113E537.

Especially, I would like to thank to my family, wife, and sons for being patient during my study. I owe acknowledgment to them for their encouragement, and love throughout difficult times in my graduate years.

EFFICIENT AND SECURE DOCUMENT SIMILARITY SEARCH OVER CLOUD UTILIZING MAPREDUCE

Mahmoud Alewiwi

Computer Science and Engineering

Ph.D. Thesis, 2015

Thesis Supervisor: Prof.Dr. Erkay Savaş

Keywords: Similarity, Privacy, Cloud Computing, MapReduce, Hadoop , Cryptography, Encryption

Abstract

Document similarity has important real life applications such as finding duplicate web sites and identifying plagiarism. While the basic techniques such as k-similarity algorithms have been long known, overwhelming amount of data, being collected such as in big data setting, calls for novel algorithms to find highly similar documents in reasonably short amount of time. In particular, pairwise comparison of documents sharing a common feature, necessitates prohibitively high storage and computation power. The wide spread availability of cloud computing provides users easy access to high storage and processing power. Furthermore, outsourcing their data to the cloud guarantees reliability and availability for their data while privacy and security concerns are not always properly addressed. This leads to the problem of protecting the privacy of sensitive data against adversaries including the cloud operator.

Generally, traditional document similarity algorithms tend to compare all the documents in a data set sharing same terms (words) with query document. In our work, we propose a new filtering technique that works on plaintext data, which decreases the number of comparisons between the query set

and the search set to find highly similar documents. The technique, referred as ZOLIP algorithm, is efficient and scalable, but does not provide security.

We also design and implement three secure similarity search algorithms for text documents, namely Secure Sketch Search, Secure Minhash Search and Secure ZOLIP. The first algorithm utilizes locality sensitive hashing techniques and cosine similarity. While the second algorithm uses the Minhash Algorithm, the last one uses the encrypted ZOLIP Signature, which is the secure version of the ZOLIP algorithm.

We utilize the Hadoop distributed file system and the MapReduce parallel programming model to scale our techniques to big data setting. Our experimental results on real data show that some of the proposed methods perform better than the previous work in the literature in terms of the number of joins, and therefore, speed.

MAPREDUCE İLE BULUT ÜZERİNDE DOKÜMANLAR İÇİN VERİMLİ VE GÜVENLİ BENZERLİK HESAPLAMA

Mahmoud Alewiwi
Bilgisayar Bilimi ve Mühendisliği
Ph.D. tez, 2015
Tez Danışmanı: Prof.Dr. Erkay Savaş

Özet

Dokümanlar arasında benzerlik arama işleminin gerçek hayatta tekrarlayan web sayfalarını ya da intihalleri bulmak gibi önemli uygulama alanıları vardır. Her ne kadar k-benzerlik algoritması gibi temel teknikler literatürde uzun zamandır mevcut olsa da, özellikle çok büyük boyutlardaki verilerle çalışmanın gerekli olduğu büyük veri uygulamalarında bu tür basit teknikler yavaş ve yetersiz kalırlar. Özellikle dokümanları ikili olarak bir ortak terimi içeriyor mu diye karşılaştırmak çok yüksek depolama ve hesaplama gücü gereksinimleri doğurur. Bulut bilişimin hızla yaygınlaşması, kullanıcıların bu ihtiyaçlarına cevap vermektedir. Veriyi bu tür bulut servis sağlayıcılar üzerinden paylaşmak, verinin erişilebilirliğini garanti etse de, verinin mahremiyeti ve gizliliği garanti edilemez. Bu durum, özellikle hassas verilerin mahremiyetini koruma problemini ortaya çıkarmıştır.

Geleneksel dokümanlar arası benzerlik bulma algoritmaları çoğunlukla sorgulanan dokümanı veri tabanındaki diğer tüm dokümanlarla karşılaştırmayı gerektirir. Bizim önerdiğimiz sistemde ise, açık (şifrelenmemiş) metin verileri üzerinde gerekli olan karşılaştırma sayısını önemli oranda azaltan yeni bir filtreleme tekniği kullanımı önerilmiştir. Bu sistem açık veriler üzerindeki ben-

zerlik karşılaştırmalarında verimli olarak çalışmaktadır ve ölçeklenebilirdir, ancak bir güvenlik sağlamaz.

Bu sistemin yanı sıra, mahremiyeti de sağlayacak üç güvenli benzerlik arama algoritması da (Secure Sketch Search, Secure Minhash Search ve Secure ZOLIP) tasarlanmıştır. Bunlardan ilki dokümanlar arasındaki kosinüs benzerliğini konum hassasiyetli özütleme (locality sensitive hashing) teknikleri kullanarak yapar. İkinci yöntem MinHash algoritmalarını kullanırken üçüncüsü ise daha önce açık metinler için tasarladığımız ZOLIP imzalarının şifrelenmiş hallerini kullanarak benzerlik hesaplaması yapar.

Önerdiğimiz yöntemleri gerçeklerken büyük veriler için de ölçeklenebilir olması için, Hadoop dağıtık dosya sistemleri ve MapReduce paralel programlama modelinden yararlanıyoruz. Gerçek veriler üzeride yaptığımız deneyler, önerilen yöntemlerin bazılarının literatürde var olan diğer sistemlerden daha az sayıda birleştirme/karşılaştırma işlemine ihtiyaç duyduğunu, ve dolayısıyla daha hızlı olduğunu göstermiştir.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

INTRODUCTION

Big data, referring to not only the huge amount of data being collected, but also associated opportunities, has big potential for major improvements in many fields from health care to business management. Therefore, there is an ever-increasing demand for efficient and scalable tools that can analyze and process immense amount of, possibly unstructured, data, which keeps increasing in size and complexity.

Finding similarities (or duplicates) among multiple data items, or documents, is one of the fundamental operations, which can be extremely challenging due to the nature of big data. In particular, similarity search on a huge data set, where the documents are represented as multi-dimensional feature vectors, necessitates pair-wise comparisons, which requires the computation of a distance metric, and therefore can be very time and resource consuming, if not infeasible.

However, complexity of establishing such a powerful infrastructure may be costly or not available especially for small and medium-sized enterprises (SMEs). Cloud computing offers an ideal solution for this problem. The currently available cloud services can provide both storage and computation

capability for massive volumes of data. This motivates us to find new and efficient document similarity searching algorithms that can work in big data setting utilizing cloud computing.

While data outsourcing to cloud is a feasible solution for many organizations the fact that the outsourced data may contain sensitive information leads to privacy breaches [2, 3]. Secure processing of outsourced data operations require protection of the confidentiality of both the outsourced data and the submitted queries. Moreover, it also requires to maintain the confidentiality of the patterns such as different accesses/queries aiming to retrieve the same data. Encryption of data prior to outsourcing may provide the confidentiality of the content of the data. However, the classical encryption methods do not provide even simple operations over the ciphertext.

In our work, we first concentrate on finding similar documents over data sets in plaintext using an efficient algorithm, which utilizes a new filtering technique and cosine similarity between two documents. Then, we propose secure search algorithms that aim to find similar documents without revealing sensitive data.

1.1 Motivations

While the basic techniques such as k-similarity algorithms have been long known, overwhelming amount of data, being collected such as in big data setting, calls for novel algorithms to find highly similar documents in reasonably short amount of time. In particular, pairwise comparison of documents' features, a key operation in calculating document similarity, necessitates prohibitively high storage and computation power.

Finding similarities (or duplicates) among multiple data items, or docu-

ments, is one of the fundamental operations, which can be extremely challenging due to the nature of big data. In particular, similarity search on a huge data set, where the documents are represented as multi-dimensional feature vectors, necessitates pair-wise comparisons, which requires the computation of a distance metric, and therefore can be very time and resource consuming, if not infeasible.

A commonly used technique, known as filtering, decreases the number of pairwise comparisons by skipping the comparison of two documents if they are not potentially similar; e.g., they do not share any common feature. Also, representation, storage, management and processing of documents play an important role in the performance of a similarity search method. A distributed file system and a parallel programming model such as MapReduce [4] are necessary components of a scalable and efficient solution in big data applications.

From security perspective, secure data mining operations require protection of the confidentiality of both the outsourced data along with its index that allows searching capability and the submitted queries. Moreover, it also requires to maintain the confidentiality of the search and access patterns such as different accesses/queries aiming the same data. Data encryption before outsourcing may provide the confidentiality of the content of the data. However, classical encryption methods do not allow even simple operations over the ciphertext. In the past few years several solutions have been proposed for efficient search operations over encrypted data utilizing a searchable index structure that accurately represents the actual data without revealing the sensitive information.

1.2 Contributions

This thesis focuses on the general problem of detecting the k-most similar documents for a given (set of) document(s). It presents four novel algorithms: i) one algorithm for unprotected document sets aiming fast and a scalable search operation based on filtering and ii) three algorithms for secure search operation utilizing various encryption techniques. In the first algorithm, where the search is performed over plaintext data, two cases are considered: i) finding k-most similar documents for each document within a given data set (*self join*), and ii) finding k-most similar documents for each document in one set from the other set (*R-S join*), for instance query set and data set. In secure search algorithms, only R-S join is considered as self join is not feasible due to the large sizes of data set used in the experiments.

The contributions of this thesis as well as the techniques employed are summarized as follows:

- We propose an efficient document similarity algorithm that search for document similarity over plaintext data sets.
- We utilize Z-order and propose a Z-order prefix filtering technique to enhance the efficiency of the algorithm.
- We utilize term frequency-inverse document frequency (tf-idf) as a term relevancy score for weight or importance of a term/word of a document.
- We use cosine similarity metric to find similarity between documents whenever it is possible.
- We also propose several approaches that enable enhanced security properties such as search and access pattern privacy and document and query confidentiality.

- We propose three secure document similarity search schemes. The first one is based on secure sketches. The second one is based on locality sensitive hashing (LSH)(i.e MinHash). The last one uses the Z-order prefix encrypted using HMAC algorithm. The security properties of the proposed algorithms are different while some of them provide access and search pattern privacy in addition to document and query confidentiality, the others provide basic security for data and query privacy.

- For all the above algorithms, we use the MapReduce parallel processing framework which is a popular computing model for big data applications in recent times.

1.3 Outline

The thesis is organized as follows: the next chapter (Chapter 2), presents a literature review on prior work related to document similarity over plain and encrypted data and indexes. In Chapter 3, we provide the preliminaries that will be used throughout the thesis. In Chapter 4, we introduce a novel document similarity search algorithm that is based on Z-order prefix filtering. Chapters 5, 6 and 7 give the details of three different secure document similarity search algorithms, respectively. In Chapter 5, we explain Secure Sketch algorithm. In Chapter 6, a secure search algorithm based on a locality sensitive hash function known as MinHash is explained. And finally, in Chapter 7 we explain the secure ZOLIP algorithm, which is the secure version of the algorithm given in Chapter 4. Finally, chapter 8 concludes the thesis.

Chapter 2

RELATED WORKS

This chapter presents a short survey on previous works in the literature related to document similarity over plaintext and encrypted documents. Efficiency and accuracy of different algorithms are discussed and their advantages and disadvantages are pointed out.

2.1 Related Work on Similarity Search

In the literature, the problem of *set-similarity* on a single machine is considered in several works [5–8]. These works are mainly focused on reducing the complexity of *vector similarity join*. Angiulli et al. [9] used the Z-order space filling curve in order to find the similarity between two high dimensional spatial data sets using Minkowski metrics. This method performs well for finding close pairs in high dimensional data, but it is not suitable for text based similarity detection. For text based similarity, as in the case of document similarity problem, the cosine similarity metric is more suitable than the Minkowski metric.

Connor and Kumar [10] suggested another technique for the similar doc-

ument detection problem. They used a binary search technique to find k-nearest neighbors (k-NN) within a selected Z hypercube. A popular approach in other works is adapting *filtering techniques* that filter out pairs that cannot surpass a given similarity threshold. Filtering decreases the number of candidates for the computation of similarity metric and, therefore, the number of similarity join operations by eliminating the documents that do not share a common important term with the query.

There are various filtering techniques used in the literature. A prefix filtering method is suggested by Chaudhuri et al. [7]. The length filtering method is utilized in the works [5] and [8]. Positional and suffix filters are proposed by Xiao et al. [11]. Sarawagi and Kirpal [6] proposed a method called *PPJoin+* that utilizes inverted index and uses a *Pair-Count* algorithm which generates pairs that share certain number of tokens. Arasu et al. [5] proposed a signature based method, in which the features of documents are represented by signatures and the similarity among the documents is calculated using the similarity of the underlying signatures. Zhu et al. [12] suggested a searching technique based on cosine similarity. They proposed an algorithm that utilizes a diagonal traversal strategy to filter out unrelated documents. In this algorithm, the elements in the data set are represented by binary vectors, meaning that only the existence of terms is considered, ignoring their frequencies or importance in the data set.

The MapReduce computing model is also considered for the similarity search problem and this leads to parallel join algorithms for large data sets that are stored on cloud servers. Elsayed et al. [13] suggested a MapReduce Model with a *Full-Filtering* technique. They used a simple filter that finds only the pairs that share common tokens. The proposed method is composed of two phases. While the first phase parses and creates the indexes for the

terms in each document, the second phase finds the similar pairs that share these terms. Vernica et al. [1] used the *PPJoin+* method [6] in order to perform the *self-join* operation. Yang et al. [14] proposed a method that uses prefix and suffix filters with two phases of MapReduce. Inverted index is used in [15] combined with prefix filtering. A modified double pass MapReduce prefix filtering method was proposed by Baraglia et al. [16]. Phan et al. [17] used Bloom filtering for building similarity pairs, in which each pair should intersect at least in one position with the arrays generated by the Bloom filters.

The previous works in the literature of similarity search do not take the importance of the terms in documents into consideration to the best of our knowledge (at least to the extent in this work). This affects the semantic similarity between documents (i.e., some documents may have the same terms but in different contexts). In our algorithm, in order to address this issue, we utilize a cosine similarity based filtering technique using the relative importance of terms in documents for finding similar documents.

Over the years, several secure similar document detection methods have been proposed in the literature. There are two main assumptions on this topic: similar document detection among two parties and similar document detection over encrypted cloud data. The core of search over cloud data depends on searchable encryption methods, therefore several different searchable encryption methods are proposed over the recent years [18, 19]

The majority of the works aim similar document detection among two parties. The parties A and B want to compute the similarity between their documents a and b respectively, without disclosing a or b. In this approach, the parties know the data of their own in plaintext form, but do not know the documents in the other party [20–22]. Jiang et al. [20] proposed a cosine

similarity based similar document detection method between two parties. They propose two approaches one with random matrix multiplication and one with component-wise homomorphic encryption. An efficient similarity search method among two parties is proposed by Murugesan et al. [21]. They explore clustering based solutions that are significantly efficient while providing high accuracy. Buyukbilen and Bakiras [22] proposed another similar document detection method between two parties. They generate document fingerprints using simhash and reduce the problem to a secure XOR operation between two bit vectors. The secure XOR operation is formulated as a secure two party computation protocol.

The other important line of research is similar document detection over encrypted cloud data. This approach is more challenging than the former one since the cloud cannot access the plaintext version of the data it stores. Wong et al. [23] propose a SCONEDB (Secure Computation ON Encrypted DataBase) model, which captures execution and security requirements. They developed an asymmetric scalar product preserving encryption (ASPE). In this method the query points and database points are encrypted differently, which avoids distance recoverability using only the encrypted values. Yao et al. [24] investigate the secure nearest neighbor problem and rephrased its definition. Instead of finding the encrypted exact nearest neighbor, server finds a relevant encrypted partition such that the exact nearest neighbor is guaranteed to be in that partition. Elmehdwi et al. [25] also consider the k-NN problem over encrypted data base outsourced to a cloud. This method can protect the confidentiality of users' search patterns and access patterns. The method uses the euclidean distance for finding the similarity and utilize several subroutines such as secure minimum, secure multiplication and secure OR operations. Overall, the method provides the same security

guarantees with the method proposed in Chapter 5 but considers Euclidean distance, where cosine similarity is considered in our work. Cosine similarity is especially useful for high-dimensional spaces. In document similarity, each term is assigned to a dimension and the documents is characterized by a vector where the value of each dimension is the corresponding tf-idf score. Therefore, cosine similarity captures the similarity among two documents better than the Euclidean similarity.

Chapter 3

PRELIMINARIES

To understand the proposed schemes and follow the pertinent discussions in this thesis, this chapter provides explanations for the following preliminaries: "Term Relevancy Scoring", "Z-Order Mapping", "Locality Sensitive Hashing" and "Hadoop and MapReduce Framework".

3.1 Term Relevancy Score

We can represent a data object (e.g., a document, an image, a video file, etc.) as a vector of features, which identifies that data object. In this thesis, we use documents that are represented by a set of terms (i.e., keywords, words from human language). More formally, each document d_i in the data set $\mathcal{D}$ contains a certain number of terms from a global dictionary T, where $|T| = \delta$ is the total number of terms in the dictionary. Each document in the data set is represented as a vector of term weights derived from the dictionary T. In our scheme, a component of the term vector for the document d_i is in fact the relevance score of the corresponding term t_j, which simply indicates the importance of the term t_j in distinguishing d_i from all the other documents

in $\mathcal{D}$.

One of the most commonly used weighting factor in information retrieval is the tf-idf value of a term in a document [26]. This factor quantifies the importance of a term in a document and combines two metrics: i) the term frequency (tf) which is the number of occurrences of the term in a document (i.e., $tf_{j,i}$ is the number of occurrence of the term t_j in the document d_i) and ii) the inverse document frequency (idf) which, represents the number of documents that contain the term t_j among the whole document set. In other words idf is a measure of the rarity of the term t_j in document set $\mathcal{D}$. The tf-idf of a term t_j in the document d_i is calculated as

$$\textit{tf-idf}_{j,i} = tf_{j,i} \times idf_j.$$

In practice, since a given term usually occurs only in a limited number of documents, the tf-idf vectors contain many zero elements and thus, tf-idf values are stored in a sparse vector to optimize the memory usage.

Let $\mathcal{S}(d_i, d_j)$ be the similarity function that quantifies the similarity between two documents, d_i and d_j. Let σ be the threshold of minimum required similarity for the pair d_i and d_j. The similarity join problem is to find the candidate d_j for the document d_i such that $\mathcal{S}(d_i, d_j) \geq \sigma$. There are different choices for suitable similarity functions depending on the application domain. The most commonly used similarity metrics in the literature for the objects d_i and d_j are described as follows

- *Jaccard similarity* $\mathcal{S}_j(d_i, d_j) = \dfrac{|d_i \cap d_j|}{|d_i \cup d_j|}$,
- *Cosine similarity* $\mathcal{S}_c(d_i, d_j) = \dfrac{d_i \cdot d_j}{||d_i|| \cdot ||d_j||}$,
- *Hamming distance* $\mathcal{S}_h(d_i, d_j) = |(d_i - d_j) \cup (d_j - d_i)|$,

which is defined as the size of their symmetric differences.

In the subsequent chapters, we use both cosine similarity and Jaccard similarity (or an approximation of the latter).

3.2 Cosine Similarity

The idea behind using cosine similarity is to take into account tf-idf values of terms in document comparison operations as the set of words with high tf-idf values are a determining factor for similarity between two documents. The cosine similarity can be calculated as

$$\mathcal{S}_c(d_i, d_j) = \frac{d_i \cdot d_j}{||d_i|| \cdot ||d_j||} = \frac{\sum_{t=1}^{\delta} d_{it} \times d_{jt}}{\sqrt{\sum_{t=1}^{\delta} d_{it}^2} \times \sqrt{\sum_{t=1}^{\delta} d_{jt}^2}},$$

where d_{it} and d_{jt} are the weights of the corresponding terms in the documents d_i and d_j. Without loss of generality, we can assume that d_i and d_j contain the same number of terms. In case there are different number of terms, we can always pad the term vector with terms whose tf-idf values are 0. From the above formula, one can understand that the terms with higher tf-idf values contribute to the cosine similarity metric significantly more than the terms with relatively smaller tf-idf values. This observation is the core of our filtering technique. Example 1 demonstrates the calculation of the similarity of documents using only the important terms.

Example 1 *Let a,b and c be documents represented with tf-idf vectors as follows abusing the notation,*

$a = (0, 8, 5, 0.25, 0.125, 0, 0.02, 0, 0, 0.1)$

$b = (0.5, 9, 4, 0, 0, 0.125, 0, 0, 0, 0)$

$c = (9, 0.2, 7, 0, 0.04, 1, 0.5, 1, 0, 7)$.

Here, let $\bar{a}, \bar{b}, \bar{c}$ *be the projected vectors using only the terms with high tf-idf values, ignoring the values less than* 1*. Then we obtain the following term vectors for the objects a, b, and c, respectively*

$\bar{a} = (0, 8, 5, 0, 0, 0, 0, 0, 0, 0)$

$\bar{b} = (0, 9, 4, 0, 0, 0, 0, 0, 0, 0)$

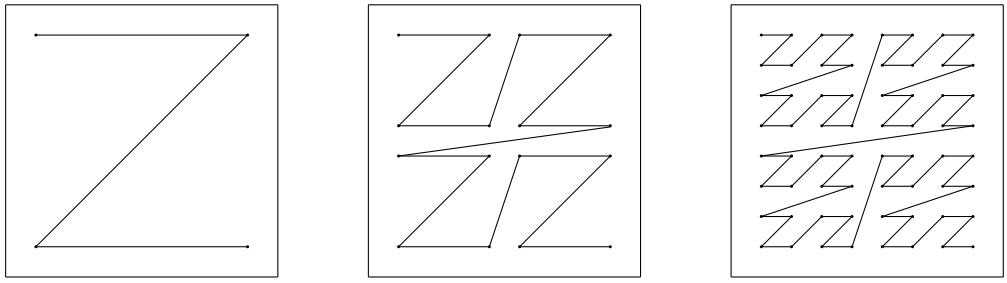
$\bar{c} = (9, 0, 7, 0, 0, 1, 0, 1, 0, 7)$.

Notice that, if we calculate the cosine similarity between each pair of the tf-idf vectors we see that $\mathcal{S}_c(\bar{a}, \bar{b}) = 0.9888 \approx \mathcal{S}_c(a, b) = 0.9883$ *and* $\mathcal{S}_c(\bar{a}, \bar{c}) = 0.2757 \approx \mathcal{S}_c(a, c) = 0.2936$*. Therefore, even though the pair* (a, c) *has more common terms, we can conclude that the closest pair is* (a, b)*, small the terms with small tf-idf values have a very low effect on the cosine similarity.*

3.3 *Z*-Order Mapping

Z-order or Morton order is a space filling curve, whose different iterations can be computed as shown in Figure 3.1. As the number of iterations increases, the space can be filled with higher accuracy. One important property of Z-order curve is that, it preserves the locality of data points in the space. Therefore, Z-order is a frequently used approach for mapping multidimensional data into one dimensional space and still supports operations such as comparison and similarity check after the mapping. Here, we formalize our terminology for the Z-order curves.

Definition 1 (Iteration on *Z*-Order Curve) *The* l^{th} *iteration of the Z-order curve in* δ*-dimensional space is a set of* 2^{δ^l} *sub-curves, where each sub-*



(a) Zeroth Iteration on Z-Order

(b) First Iteration on Z-Order

(c) Second Iteration on Z-Order

Figure 3.1: Z-Order Space Filling

curve is composed of points whose coordinates have the same l most significant bits.

Figure 3.1 illustrates the Z-order sub-curves for different iterations on the original curve.

Definition 2 (Z-Shape of Order l) *A Z-shape of order l in δ-dimensional space is any of the Z-order sub-curves in an l^{th} iteration of the Z-order curve.*

Intuitively, each sub-curve in an iteration on a Z-order curve is a Z-shape. For instance, the circles labeled as A and B in Figure 3.2 enclose the second and first order Z-shapes, respectively.

Definition 3 (Z-Value) *The Z-value of a data point in the multidimensional space is obtained by interleaving the bits of binary representation of the data point coordinate values.*

Points in the δ-dimensional space are represented with scalars (i.e., Z-value), which preserve their locality in such a way that similarity and comparison between points can be calculated. For instance, the point $(110, 001)$ in the Z-shape of order 2 in Figure 3.2 is mapped to Z-value of 101001. In summary,

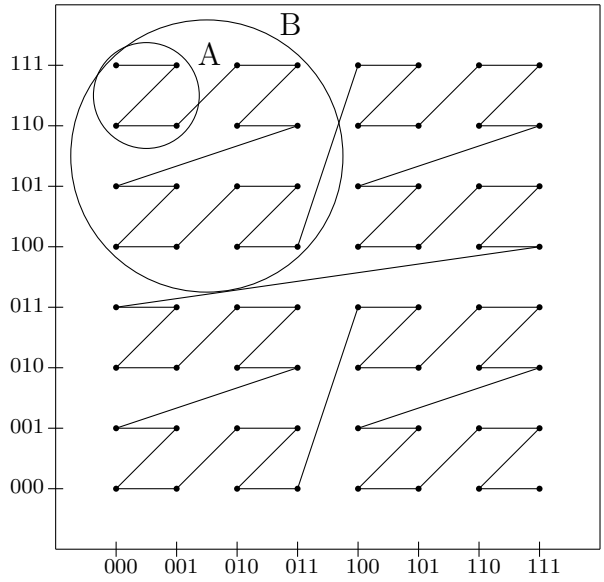


Figure 3.2: Data Points on Z-Order Curve

the Z-value of a point in multidimensional space is simply a scalar that can be used in various applications. In the literature [10, 27, 28], the Z-value is used to find the k-nearest neighbors in spatial data sets.

A document represented by a vector of tf-idf values can be viewed as a point in the multidimensional space of terms. Then, the Z-order mapping can be used to map a document into one dimensional space by preserving the locality of the document in the multidimensional space. Consequently, it will be possible to compute the similarity of documents using their Z-values.

For instance, in the two-dimensional space in Figure 3.2 (i.e., $\delta = 2$), the circle A denotes a Z-shape of the second order while the one denoted by B is a Z-shape of the first order. The points in the Z-shape A are $(000, 110)$, $(000, 111)$, $(001, 110)$, and $(001, 111)$. Their corresponding Z-values, namely (010100), (010101), (010110), and (010111), have the same prefix of (0101). Similarly, the Z-values of the points in B share the prefix (01). From this, we

can conclude that the points on a Z-shape of larger order (i.e., which share a longer prefix) are closer. The common prefix in Z-values of the points can be used to calculate the similarity of documents (i.e., closeness in the multidimensional space), where the coordinates of the points are the tf-idf values of the corresponding terms in the documents.

The next example demonstrates a technique that uses the Z-order mapping to obtain the most important terms in a document. Let $\lambda \cdot \delta$ be the number of prefix bits shared in the same Z-shape, where δ is the number of terms in the dictionary T (i.e., the dimension of the document space) and λ is an accuracy parameter chosen appropriately.

Example 2 *Recall that in Example 1, we have the following term vectors for three documents, where* $\delta = 10$

$a = (0, 8, 5, 0.25, 0.125, 0, 0.02, 0, 0, 0.1)$

$b = (0.5, 9, 4, 0, 0, 0.125, 0, 0, 0, 0)$

$c = (9, 0.2, 7, 0, 0.04, 1, 0.5, 1, 0, 7)$.

In order to represent all the tf-idf values, we need four and three bits to represent their integer and fractional parts, respectively. Thus, we need 7 *bits in total for the tf-idf values. However, by setting* $\lambda = 3$ *at the expense of losing precision, we get the prefix values of the term vectors shown in the following table.*

Doc a

Z-Order iteration	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
1^{st} *iteration prefix*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
2^{nd} *iteration prefix*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
3^{rd} *iteration prefix*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Doc b

Z-Order iteration	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
1^{st} *iteration prefix*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
2^{nd} *iteration prefix*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
3^{rd} *iteration prefix*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Doc c

Z-Order iteration	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
1^{st} *iteration prefix*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
2^{nd} *iteration prefix*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
3^{rd} *iteration prefix*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*

Notice that documents a and b have the same prefix for all three iterations.

This is natural since a and b are very similar as shown in Example 1.

In Chapter 4, we first develop an efficient method to find similar documents such that, cosine similarity between two documents is computed only if they are in the same Z-shape of the λ-th order; i.e., having the same $\lambda\delta$ bits as prefix in their Z values. On the other hand, this method, which eliminates the need of calculating the cosine similarity between many dissimilar document pairs and therefore yields a very efficient implementation, can only be applicable in cases where highly similar documents exist. If the data set does not contain sufficiently many (i.e., k) highly similar documents, it is required to also consider the documents that do not reside in the same Z-shape of λ order.

For similar documents that do not reside in the same Z-shape of the desired order, we propose a slightly different method, in which only documents that contain common important terms (i.e., that have high tf-idf values) will be compared. In other words, if two documents contain at least one important term in common, their cosine similarity is calculated, otherwise the computation is skipped.

Definition 4 (l-th Iteration Projection) *Let $d_i = (d_{i_1}, \ldots, d_{i_\delta})$ be a document represented in δ-dimensional space of term tf-idf values. Let also $d_{i_j}^l$ denote the most significant l-bits of the values d_{i_j} for $j = 1, \ldots, \delta$. Then we can define the projection of the vector d_i to l-th iteration on the Z-order curve as*

$$\bar{d}_{i_j}^l = \begin{cases} d_{i_j}, & \text{if } \; d_{i_j}^l > 0 \\ 0, & \text{otherwise,} \end{cases}$$

for $j \in \{1, \ldots, \delta\}$.

The projection takes already a sparse vector d_i and generates an expectedly much sparser vector of term tf-idf values, $\bar{d}_i$. We check the new vector, having only the important terms as non-zero elements, to see whether it represents the document with a sufficiently high accuracy.

In the proposed method, we start with 1-*st* iteration projection $\bar{d}_i^{\,1}$ of document, for which we try to find similar documents, and compute the similarity, $\mathcal{S}_c(d_i, \bar{d}_i^1)$. If the similarity is larger than a predefined similarity threshold σ, namely $\mathcal{S}_c(d_i, \bar{d}_i^1) > \sigma$, then we use 1-*st* iteration projections of the two documents to decide to compute their cosine similarities.

If $\mathcal{S}_c(d_i, \bar{d}_i^1) < \sigma$, then we use a higher level projection $\bar{d}_i^{\,l}$ with $l > 1$, where l is the minimum value that satisfies the threshold σ. Then, we compute $\mathcal{S}_c(d_i, d_j)$ of two documents d_i and d_j only if $\bar{d}_i^l$ and $\bar{d}_j^l$ have at least one common non-zero term. The following example illustrates the proposed technique.

Example 3 *Let the threshold and the precision parameters be set as,* $\sigma = 0.8$ *and* $\lambda = 3$*, respectively. Also let the data set has the following three documents with* $\delta = 13$*,*

$a = (0, 27, 17, 0, 5, 9, 0, 11, 6, 11, 0, 13, 14)$

$b = (0, 27, 21, 0, 0, 0, 15, 0, 5, 0, 0, 6, 10)$

$c = (0, 0, 0, 29, 0, 0, 16, 0, 0, 0, 4, 0, 5)$.

Using the first and second iteration projections, we can obtain the following vectors for the document a,

$$\bar{a}^1 = (0, 27, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

$$\bar{a}^2 = (0, 27, 17, 0, 0, 9, 0, 11, 0, 11, 0, 13, 14)$$

The corresponding cosine similarities are computed as

$$\mathcal{S}_c(a, \bar{a}^1) = 0.7590,$$
$$\mathcal{S}_c(a, \bar{a}^2) = 0.8955.$$

As can be observed, the second iteration projection $\bar{a}^2$ *satisfies the given threshold. Therefore, documents whose* 2*nd iteration projections do not share any term with* $\bar{a}^2$ *(i.e., that have a zero tf-idf score for the corresponding terms that appear in* $\bar{a}^2$*), will be filtered out and their cosine similarities will not be computed. Note that,* $\bar{a}^2$ *has seven nonzero terms as opposed to nine nonzero terms in the original vector* a*, which potentially eliminates unnecessary similarity comparisons.*

Here, $\mathcal{S}_c(b, a)$ *is computed since* $\bar{b}^2$ *and* $\bar{a}^2$ *have common terms. However,* $\mathcal{S}_c(c, a)$ *is not computed since* $\bar{a}^2$ *and* $\bar{c}^2$ *do not share any common term. Although* a *and* c *have a common term (i.e., the last term), it is omitted due to its low tf-idf value in* c*. Indeed, the document* b *is much closer to* a *as the following cosine similarities of the original documents indicate*

$$\mathcal{S}_c(a, b) = 0.8045,$$
$$\mathcal{S}_c(a, c) = 0.0493.$$

The selected λ *value which is used to improve the accuracy is a data set dependent, and should be determined experimentally.* The methods that are briefly introduced in this section, will be formalized in Chapter 4.

3.4 Locality Sensitive Hashing (LSH)

The main principle of locality sensitive hashing is to represent arbitrary length features of data items in constant sized sets that are called signatures. The idea is to hash each feature set F_i into a constant size (and preferably small) signature that can represent the similarity accurately. Signatures provide an approximation for measuring the similarity between two data items and the accuracy of the approximation is directly related with the length of the signatures such that, the longer the signature the more accurate the result. However, while very small signatures are sufficient for detection of either almost identical or totally unrelated stuff, relatively longer signatures are required for similarities in between.

The goal of LSH functions is that, for inputs with high similarity, the hash functions should provide the same output with high probability and provide different output with high probability otherwise. Note that, this principle is completely different from the principle of cryptographic hash functions, where finding two different inputs that provide the same output is very difficult.

The signatures are represented as sets. A well known metric for representing the similarity between two sets is the Jaccard similarity.

Definition 5 (Jaccard Similarity) *Let A and B be two sets, the Jaccard similarity of A and B is defined as in Equation* (3.1)*.*

$$J_s(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{3.1}$$

The elements of signatures are constructed using MinHash functions [29] which is defined as follows.

Definition 6 (MinHash) *: Let Δ be a set of elements, P be a permutation on Δ and $P[i]$ be the i^{th} element in the permutation P. MinHash of a set $D \subseteq \Delta$ under permutation P is defined as:*

$$h_P(D) = min(\{i | 1 \leq i \leq |\Delta| \wedge P[i] \in \mathcal{D}\})$$

Each data element signature is generated by λ MinHash functions each applied with a different randomly chosen permutation. The resulting signature for a data set element D is:

$$Sig(D) = \{h_{P_1(D)}, \ldots, h_{P_\lambda}(D)\}, \tag{3.2}$$

where h_{P_j} is the MinHash Function under permutation P_j.

The MinHash functions are used while generating the signatures since there is a perfect correlation between the Jaccard similarity and MinHash functions. The probability that MinHash functions provide the same output for two inputs A and B is equal to the Jaccard similarity between A and B as shown in the Equation (3.3).

$$Pr[h(A) = h(B)] = J_s(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{3.3}$$

As MinHash functions with different permutations provide independent experiments, using longer signatures (i.e., larger λ) provides more accurate results.

3.5 Hadoop and MapReduce Framework

Currently, the MapReduce programming model became a common model for parallel processing. MapReduce employs a parallel execution and coor-

dination model that can be used to manage large-scale computations over massive data sets [29, 30]. The Hadoop framework [31] is a well known and widely used MapReduce parallel processing framework. It works on a cluster of computers called cloud. The Hadoop framework works utilizing the MapReduce programing modeling.

The Hadoop framework contains two main parts, namely Hadoop Distributed File System (HDFS) and NextGen MapReduce (YARN). The files are stored in the HDFS, which is a special distributed file system. YARN or MapReduce 2.0 is a system that facilitates writing arbitrary distributed processing frameworks and applications of large data sets.

Data replication is one of the key factors that improves the effectiveness of the Hadoop framework, that survives node failures while utilizing huge number of cluster nodes for data intensive computations. The MapReduce model is successfully implemented in the Hadoop framework as the details of the network communication, process management, interprocess communication, efficient massive data movement and fault tolerance are transparent to the user. Typically, a developer needs to provide only configuration parameters and several high-level routines.

The Hadoop framework is used by most of the major actors including Google, Yahoo and IBM, largely for applications involving search engines and large-scale data processing (e.g., big data applications).

The MapReduce model is based on two functions, *Map* and *Reduce*. The Map function is responsible for assigning a list of data items, represented as key-value pairs to cluster nodes. The Map function receives key-value pairs, and sends the result as intermediate data to the reducer. The Reducer function gets the mapped data and applies the processing operation. It receives the intermediate data as a key and a list of values as (key,[values]).

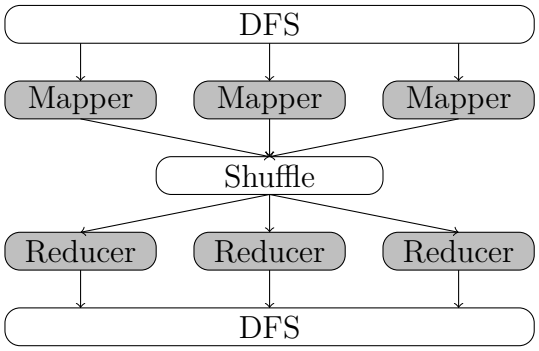


Figure 3.3: MapReduce Job Execution

The signature of these two functions are

$$\text{map:}(k_1,v_1) \rightarrow [(k_2,v_2)]$$
$$\text{reduce:}(k_2,[v_2]) \rightarrow [(k_3,v_3)].$$

The shuffling process, between the Map and Reduce functions as illustrated in Figure 3.3, is responsible for designating all keys with the same value to the same computation node. The reducer do the desired operations for records sharing a common property and send the final result to user. Figure 3.3 shows the working principle of MapReduce.

3.6 Hash-based Message Authentication Code (HMAC)

Hash-based message authentication code known as HMAC is one of the popular deterministic hashing functions used in cryptography [32]. It is used for constructing a fixed size message authentication code using a secret cryptographic key. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying construction (e.g. a cryptographic hash function), the lengths of its output, and the secret key. In

Chapters 6 and 7, we use SHA-based HMAC functions for the document signatures. HMAC function can be calculated using the following formula

$$HMAK(K, m) = H((K \oplus opad)||H((K \oplus ipad||m)),$$

where H can be a cryptographic hash function, *opad* is the outer padding, *ipad* is the inner padding, K is a secret key appropriately padded, and m is the message, data or document.

Chapter 4

EFFICIENT DOCUMENT SIMILARITY SEARCH UTILIZING Z-ORDER PREFIX FILTERING

This chapter proposes a new, efficient document similarity search algorithm [33]. The algorithm uses a new document filtering technique utilizing the prefixes obtained via Z-order space-filling curves as explained previously. The prefix in this algorithm filters the documents that do not share only important terms. The subsequent sections in this chapter describe the algorithm and present comparison with another technique in the literature. The proposed algorithm shows a desired improvement in the time performance. Last section contains accuracy evaluation for the new algorithm.

4.1 Introduction

There is an ever-increasing demand for efficient and scalable tools that can analyze and process immense amount of, possibly unstructured, data, which keeps increasing in size and complexity. Finding similarities (or duplicates) among multiple data items, or documents, is one of the fundamental operations, which can be extremely challenging due to the nature of big data. In particular, similarity search on a huge data set, where the documents are represented as multi-dimensional feature vectors, necessitates pair-wise comparisons, which requires the computation of a distance metric, and therefore can be very time and resource consuming, if not infeasible.

A commonly used technique, known as filtering, decreases the number of pairwise comparisons by skipping the comparison of two documents if they are not potentially similar; e.g., they do not share any common feature. Also, representation, storage, management and processing of documents play an important role in the performance of a similarity search method. A distributed file system and a parallel programming model such as MapReduce [4] are necessary components of a scalable and efficient solution in big data applications.

This work focuses on the general problem of detecting the k-most similar documents for each document within a given data set (henceforth *self join*), and between two arbitrary sets of documents (R-S join), namely query set and data set. The problem is formalized as follows.

Definition 7 (R-S Join Top-k Set Similarity) *Let $\mathcal{D}$ be a set of documents $\{d_1, \ldots, d_n\}$, $d_i \in \mathcal{D}$. Let $\mathcal{Q}$ be a set of query documents $\{q_1, \ldots, q_m\}$, $q_j \in \mathcal{Q}$. Then R-S top-k set similarity is defined as:*

$$\forall q_j \in \mathcal{Q}, \textit{top-k}(q_j, \mathcal{D}) = \{d_{j1}, \ldots, d_{jk}\},$$

where d_{ji} is the i^{th} nearest record to q_j in $\mathcal{D}$.

Definition 8 (Self Join Top-k Set Similarity) *Let $\mathcal{D}$ be a set of documents $\{d_1, \ldots, d_n\}$, $d_i \in \mathcal{D}$. Then self join top-k set similarity is defined as:*

$$\forall d_j \in \mathcal{D}, top\text{-}k(d_j, \mathcal{D}) = \{d_{j1}, \ldots, d_{jk}\},$$

where $d_{ji} \neq d_j$ is the i^{th} nearest record to d_j in $\mathcal{D}$.

Intuitively, the self join case is the generalization of the R-S join case such that $\mathcal{Q} = \mathcal{D}$.

The trivial solution for the set similarity problem, for two sets of items $\mathcal{Q}$ and $\mathcal{D}$, is to compare each element in Q with each element in $\mathcal{D}$. This solution has $\mathcal{O}(\delta mn)$ complexity, where m and n are the number of elements in $\mathcal{Q}$ and $\mathcal{D}$ respectively, and δ is the number of dimensions (i.e., features) in $\mathcal{Q}$ and $\mathcal{D}$. Recent trends and research are concerned about computing set similarity-join algorithms in an efficient and high performance manner, hence new set similarity-join algorithms that reduce the number of comparisons are proposed.

Current research mainly adapts *filtering techniques* that filter out pairs that have similarity below a given threshold. Clearly, the adopted filtering technique plays the utmost role in the efficiency as well as the effectiveness of a similarity search algorithm. In this work, we propose a new cosine similarity based filtering technique to improve the performance of the similarity calculation.

In our solution, we suggest a new Z-order based filtering technique in order to eliminate dissimilar documents before performing the costly operation of calculating the cosine similarity. Documents in a data set are represented as points on Z-order space filling curves on multidimensional

space of documents' features. A projection based on the most important features of documents is utilized to filter out dissimilar documents that do not share common important features. The proposed method also filters out documents that only share features of low importance, which have very little effect on the similarity between the data objects. The technique provides a remarkable improvement, especially on finding highly similar documents very quickly. And finally, we describe the algorithms to implement the proposed technique as parallel programs that can take advantage of the Hadoop [31] MapReduce parallel programming framework.

The rest of the chapter is organized as follows. The details of the proposed Z-order with Lambda Iteration Prefix (ZOLIP) filtering algorithm are explained in Section 4.2. Section 4.3 presents the experimental results that show the accuracy and the efficiency of the proposed method. Finally, the conclusions are provided in Section 4.4.

4.2 The Proposed Filtering Method

In this section we explain the proposed techniques of the Z-order with Lambda Iteration Prefix (ZOLIP) filtering algorithm. The algorithm has three phases. In the first phase, the ZOLIP algorithm considers the document vectors that lie on the same Z-shape of λ order in the Z-order curve. In other words, this phase is designed to find highly similar (i.e., near duplicate) documents in the data set. If this phase cannot find k documents, the second phase considers the documents that are on different Z-shapes. In the second phase, the algorithm finds the most important terms in the queried document such that any document that does not contain any of those important terms, cannot be in the top k similar list of the queried document. Then the similarities

with the documents sharing important terms are calculated. Finally, the last phase performs the actual join operations such that the outputs of the first and the second phases are combined and sorted to find the top k similar list of the queried document.

4.2.1 Phase 1: Near-Duplicate Detection (NDD)

Initially, the first phase reads each document record that is composed of a key/value pair. While the key represents a unique document id, the value holds a vector which contains the tf-idf values of the terms of the corresponding document. Instead of the classical vectors, sparse vector representation is used for improving the performance and easing memory requirements as most of the vector elements are zero.

Sparse vector is a data structure that stores only the none-zero entries together with corresponding index values. We assume that the number of dimensions (δ), the number of iterations on the curve (λ) and the maximum tf-idf value (μ) in the whole data set are publicly known and passed as a configuration parameter to the map function in the proposed method, which is described in Algorithm 1. Note that, the operations over sparse vectors are optimized even though the algorithm descriptions are given in classical forms. For instance, the `for` loop in Steps 4-6 in Algorithm 1 is only executed for non-zero tf-idf values.

The map function in Algorithm 1 receives the tf-idf scores of each document in the data set $\mathcal{D}$, its id, $docId$, the maximum tf-idf value μ, the dimension δ, and the precision λ for tf-idf values.

The "bitLength" function (Step 2) returns the bit length of the binary representation for the maximum tf-idf value (μ). Then, the method goes over the coordinate values and normalizes them so that they are represented

Algorithm 1 Near-Dupplicate Detection(NDD)

1: **procedure** MAP($docId, d, \mu, \delta, \lambda$)
2: $\gamma \leftarrow bitLength(\mu)$
3: $ZOLIP \leftarrow null$
4: **for** $j = 1$ to δ **do**
5: $d[j] \leftarrow Normalize(d[j], \gamma)$
6: **end for**
7: **for** $l = 1$ to λ **do**
8: **for** $j = 1$ to δ **do**
9: $bit \leftarrow getBit(d[j], l)$
10: $ZOLIP \leftarrow ZOLIP||bit$
11: **end for**
12: **end for**
13: $sendToReducer(ZOLIP, \langle docId : d \rangle)$
14: **end procedure**

▷ $listDOC$ contains the documents with the same $ZOLIP$ key
15: **procedure** REDUCE($ZOLIP$, $\{\langle docId : d \rangle\}$)
16: **for all** $d_i \in listDOC$ **do** ▷ for each document in $listDOC$
17: $ksim_i \leftarrow null$
18: **for all** $doc_j \in listDOC$ and $i \neq j$ **do**
19: $ksim_i \leftarrow findTKSD()$ ▷ compute similarity and sort
20: **end for**

21: **end for**
22: **return** $\forall i$ $(\langle docId_i : d \rangle, \langle \{\langle ksim_i, \mathcal{S}_c \rangle\}, (yes/no) \rangle)$
23: **end procedure**

with the number of bits used for the maximum tf-idf value (Step 5). The "Normalize" function gets the bit length of each coordinate tf-idf value and finds the difference (dif) between the number of bits of the maximum tf-idf and the tf-idf of the current coordinate, and pads (dif)-many zero bits to the left of the actual coordinate value to make all the tf-idf values represented with the same number of bits.

Then, the map function extracts the bits of the normalized tf-idf value of each term starting from the most significant bit using the function "getBit" (Step 9), which returns the l-th most significant bit. This operation will be repeated for each of the λ bits of the tf-idf scores. At each iteration, the mapper interleaves the extracted bit and concatenates it to a $ZOLIP$ key (Step 10), which is a sparse vector that represents the none-zero bits in the Z-value.

The key/value pair sent to the reducer is composed of the $ZOLIP$ key and the list of document identifiers and their tf-idf scores as in

$(ZOLIPkey, \langle docId : d \rangle)$.

All the documents that have the same $ZOLIP$ key are grouped by the shuffle process and sent to the reducer instances.

The reducers receive the list of documents that have the same ZOLIP key and calculates the cosine similarity between these records to find the top-k similar documents for each document. The function "findTKSD" (which stands for "find top-k similar documents") calculates the similarity between documents and sorts the documents depending on their similarity. Then, it selects the top-k similar documents and saves them in a sorted list "$ksim_i$".

Figure 4.1 illustrates an example, in which the mapper creates the ZOLIP keys Z_1, for documents "a" and "b" and Z_2 for document "c". Then, it passes the key value pairs to the reducer. The reducer receives the documents with

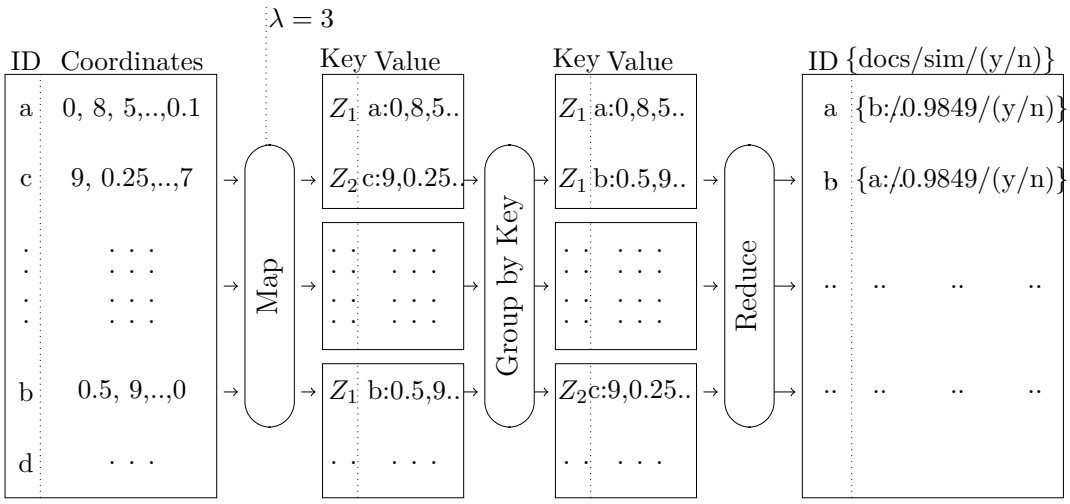


Figure 4.1: An Example Execution of ZOLIP Phase 1

the same ZOLIP key, and compares these documents to find the top-k similar documents.

The reduce function returns the records in the format

$(\langle docId_i : d\rangle, \langle\{\langle ksim_i, \mathcal{S}_c\rangle\}, (yes/no)\rangle)$.

Here, the first element is the key of the record (i.e., the identifier of the document that we intent to find its k-similar documents). The second element, which is the value of the record, contains two parts. The first part is a list that contains the top k-similar document identifiers with their corresponding similarity values as $(\mathcal{S}_c)$. The second part (i.e., (yes/no)) is an indicator that shows whether the top-k similar documents are found or not. If k similar documents are found in this step, no further search is applied for this document. Otherwise, further search in a different Z-shape is required.

Algorithm 1 represents the pseudo code based on the MapReduce programming model. The algorithm can be used both for self-join and R-S join operations. Algorithm 1 is described for self-join operation, in which we find k similar documents for every document in the data set. In R-S join operation, documents should be labeled as query documents and data set

documents. Consequently, in Step 16 of Algorithm 1 only documents in the query set should be considered while Step 18 should use the documents in the data set.

The NDD phase is expected to find k similar documents for a relatively large number of documents without further computations needed for them. While this reduces the overall complexity of the top-k similarity search, further computation is needed for the documents, for which k similar documents are not found. To this end, we propose a method based on common important terms to complete the computation for the remaining part of the database.

4.2.2 Phase 2: Common Important Terms (CIT)

Some documents, although similar, may reside in different Z-shapes. In order to calculate the similarity between two such documents, we stipulate that they share at least one important (i.e., high tf-idf valued) term. We, therefore, propose utilizing only the most important terms to eliminate similarity computation for the distant documents. Note that the probability of two documents, that do not share any important term, being in the top-k similar documents of each other is very low.

The common important terms (CIT) phase contains two mapper and one reducer functions. The first mapper function reads the output of the NDD phase to check the (yes/no) value for a document. If k similar documents are not found for the document, the second phase, common important terms (CIT), utilizes the l-th iteration projection in Z order curves to determine the important terms in the document that will be used to filter out dissimilar documents.

The first mapper (the procedure MAP1() in Algorithm 2) computes the projection $\bar{d}_i^l$ of document d_i, which contains tf-idf values of the most impor-

Algorithm 2 Common Important Terms(CIT)

1: **procedure** MAP1($docId, \langle\{\langle ksimilar, \mathcal{S}_c\rangle\}, (yes/no)\rangle, \lambda, \mu$)
2: ▷ this mapper reads the output of phase 1
3: **if** $(yes/no) = no$ **then**
4: $\bar{d}_i^0 \leftarrow createEmptyVec()$
5: $sim \leftarrow 0$
6: **for** $l = 1$ to λ AND $sim < \sigma$ **do**
7: ▷ λ is the number of bits in tf-idf scores
8: $\bar{d}_i^l \leftarrow Project(d_i, l)$
9: $sim \leftarrow \mathcal{S}_c(d_i, \bar{d}_i^l)$
10: **end for**
11: **for** $j = 1$ to δ **do**
12: **if** $\bar{d}_{ij}^l \neq 0$ **then**
13: $term \leftarrow \bar{d}_{ij}^l$
14: sendToReducer($term, \langle Q : docId, d_i\rangle$)
15: **end if**
16: **end for**
17: **end if**
18: **end procedure**

19: **procedure** MAP2($docId, d_i$)) ▷ It reads the data set documents and functions exactly as the first mapper.
20: sendToReducer($term, \langle D : docId, d_i\rangle$)
21: **end procedure**

```
22: procedure REDUCE(term,{⟨(Q/D) : docId, d_i⟩})
                    ▷ Documents will be partitioned into query and data sets
23:     QList ← null
24:     DList ← null
25:     for all d_i do
26:         if d_i is a query document then
27:             addQList(d_i)
28:         else
29:             addDList(d_i)
30:         end if
31:     end for
32:     for all d_i ∈ Qlist do
33:         candidates_i ← null
34:         for all d_j ∈ Dlist do
35:             candidates_i ← docIdD_j
36:         end for
37:         for d_j ∈ candidates_i do
38:             return (⟨docId_i : docId_j⟩ , S_c(d_i, d_j))
39:         end for
40:     end for
41: end procedure
```

tant terms in the document. Then, the similarity between $\bar{d}_i^1$ and the original vector d_i will be calculated. If the similarity threshold is satisfied, namely $\mathcal{S}_c(d_i, \bar{d}_i^1) \geq \sigma$, the terms in $\bar{d}_i^1$ are sent to the reducer. Otherwise, a higher degree projection $\bar{d}_i^l$ is calculated to satisfy the threshold.

Here, MAP1() creates a key-value pair for each important term, in which the term is used as a key, and the vector d_i and the document id are considered as the value of the pair. In other words, the original record for the document is replicated as many times as the number of important terms in the projection $\bar{d}_i^l$. In order to distinguish the documents in the query set from the documents in the data set (the latter will be processed by the second mapper MAP2() function), a prefix "Q:" is added in the value part of the pair. The resulting record is of the format, $(term, \langle Q : docId : d_i \rangle)$.

Using the setting given in Example 3, Figure 4.2 illustrates an example execution of the CIT algorithm steps for finding similar documents that have common important terms. Note that, in the mapper stage, the term indices that are exposed to the reducer correspond only to the terms with high importance (i.e., the terms that appear in $\bar{a}^2$).

The second mapper, (the procedure MAP2() in Algorithm 2), performs the same operations as the first mapper, but this time on the data set documents, with the only difference that, the prefix "D:" is added instead of "Q:". The resulting record format is of the form $(term, \langle D : docID : d_i \rangle)$. As in the case of the document in the query set, MAP2() creates as many new records as the number of important terms for the corresponding document.

The reducer in this phase receives the key value pairs from the first and the second mappers. Then, it partitions the records into two sets, $Qlist$ and $Dlist$, depending on whether the document is in the query or in the data set. The first mapper returns the important terms in the query documents com-

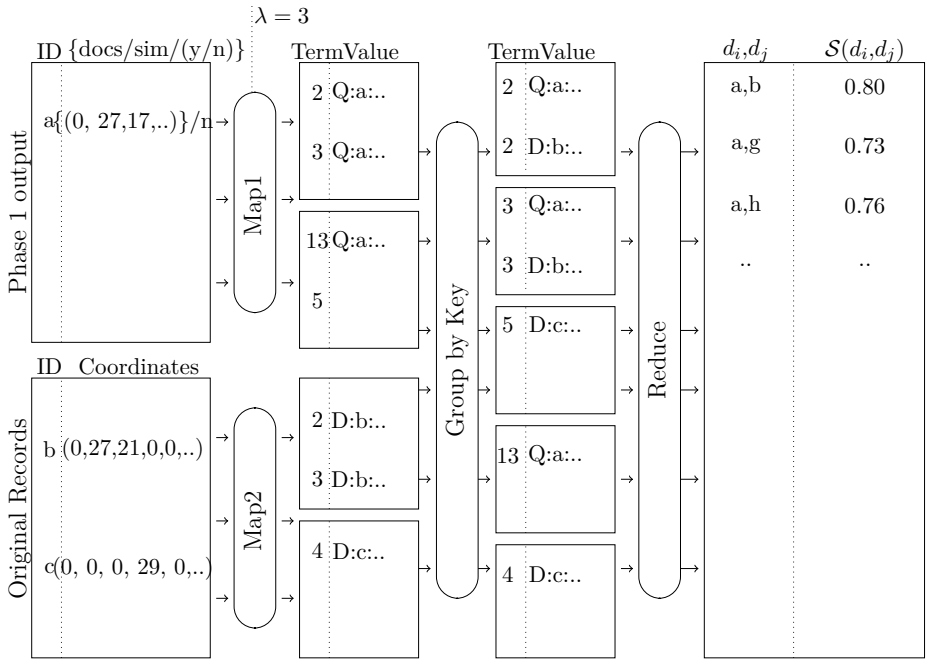


Figure 4.2: An Example Execution of ZOLIP Phase 2

bined with their coordinates, while the second mapper returns the important terms in data set documents combined with their coordinates. The reducer creates two partitions. The partition can easily be done by checking the prefixes in the value part, Q and D. Consequently, for every document in the query and data set, the reducer stores them in $Qlist$ and $Dlist$, respectively.

The last stage of the reducer calculates the similarity between $d_i \in Qlist$ and $d_j \in Dlist$. Then, the identifiers of d_i and d_j are combined to form a key for the pair in which the value is $\mathcal{S}_c(d_i,d_j)$. Consequently, the final output of this phase is of the form, $(\langle docId_i : docId_j \rangle,\ \mathcal{S}_c(d_i,d_j))$, where d_i and d_j represent the documents in the query set and the data set, respectively. Figure 4.2 visualizes all the steps on the setting from Example 3.

4.2.3 Phase 3: Join Phase(JP)

In the join phase (steps are described in Algorithm 3), in order to find the actual top-k similar documents, the join operation is applied on the results from the first and the second phases. This phase contains two mappers. The first mapper receives the output from the first phase and sends one key-value pair to the reducer for each document in the list $ksim_i$, in which $docId$ is the key and the tf-idf score vector of the document in $ksim_i$ is the value. Thus, the output of the first mapper is of the form, ($docIdQ$,$\langle docIdD : \mathcal{S}_c \rangle$).

The second mapper also works in a similar way. It receives the pair, where the key is the similar document identifiers $\langle docIdD, docIdQ \rangle$ and the value is the corresponding similarity. The mapper parses the $docIdQ$ and $docIdD$. After this step, it exposes the $docIdQ$ as a key and $\mathcal{S}_c$ prefixed with $docIdD$ as a value. The output of this mapper is also of the form, ($docIdQ$, $\langle docIdD : \mathcal{S}_c \rangle$).

The last step in the algorithm is to get all the candidate documents and select the top-k similar documents among them. The reducer receives the $docIdQ$ key from the previous mappers and a list of documents that contain the same key (i.e term) as the value. Here, the list is sorted and the first k documents with the highest similarity values are considered as the top-k similar documents. The final result of this reducer is of the form, ($docIdQ, \{\langle docIdD : \mathcal{S}_c \rangle\}$). Figure 4.3 visualizes the Join phase of the outputs of Examples 2 and 3, where the details are given in Algorithm 3.

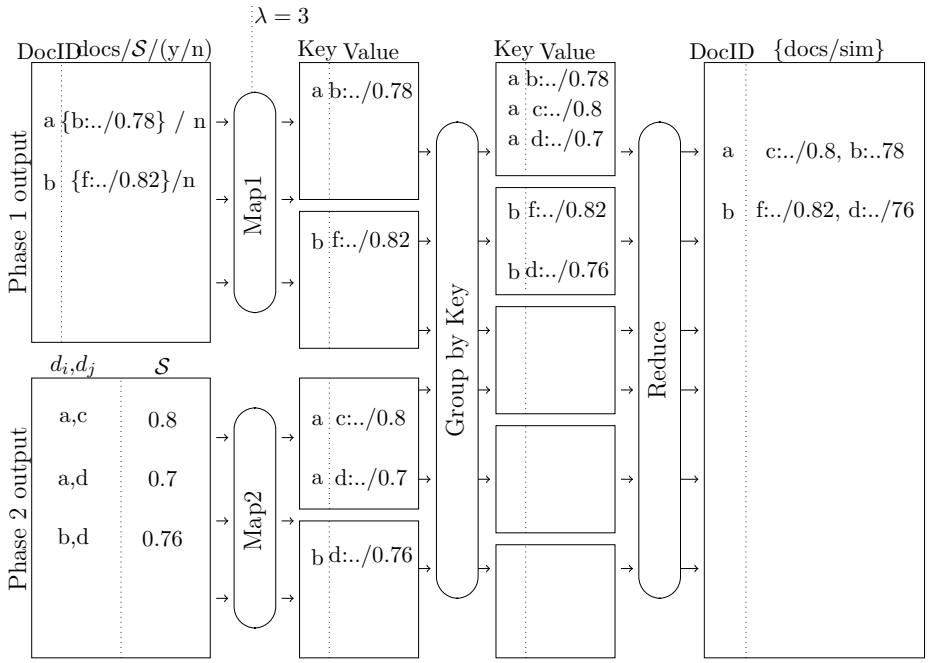


Figure 4.3: An Example Execution of ZOLIP Phase 3

Algorithm 3 Join Phase

procedure MAP1($docId, \langle\langle ksim, \mathcal{S}_c\rangle, (yes, no)\rangle$)
▷ this mapper will reads phase 1 output
$docIdQ \leftarrow docId$
for all $docIdD \in ksim$ **do**
$sendToReducer(docIdQ, \langle docIdD : \mathcal{S}_c\rangle)$
end for
end procedure
procedure MAP2($\langle docIdQ : docIdD\rangle, \mathcal{S}_c$))
▷ this mapper will reads phase 2 output
$sendToReducer(docIdQ, \langle docIdD : \mathcal{S}_c\rangle)$
end procedure
procedure REDUCE($docIdQ, \{\langle docIdD, \mathcal{S}_c\rangle\}$)
$topK \leftarrow$ Sort($docIdD, \mathcal{S}_c$)
expose(docIdQ,topKSimilar)
end procedure

4.2.4 R-S Join

The R-S join is used to find k most similar documents for a batch of query documents, which are not a part of the data set. The algorithms given above are described mostly for self join operation that finds top k similar documents for every document from the same data set. In order to adapt the proposed algorithms for R-S join operations, we need to label documents with "Q:" and "D:". Also, several simple modifications are necessary to the algorithm steps. As the modifications are only minor and straightforward, we do not include separate algorithms for R-S join operations.

The time complexity of the R-S join depends on the number of documents in the query set and an R-S join usually requires much less time than the self join since the number of the queried documents is expected to be much lower than the number of documents in the data set. We analyze the effect of the query size by testing the algorithm on queries with various number of documents in the next section.

4.3 Experiments

In this section, we first describe the system model and the data sets used in our experiments. Then, we compare the proposed method with the method by Vernica et al. [1]. The proposed method is tested with different parameter settings to show its efficiency and scalability in the big data context. Finally, we discuss the accuracy of our method and the effect of increasing λ parameter on its accuracy and efficiency.

4.3.1 Setup and Data Description

We used the Cloudera CDH4 installation with a 3-node cluster: Two nodes with Xeon processor E5-1650 3.5 GHz of 12 cores, 15.6 GB of RAM and 1 TB hard disk; and one node with Core i7 3.07 GHz of 8 cores, 15.7 GB of RAM and 0.5 TB hard disk.On each node, Ubuntu 12.04 LTS, 64 -bit operating system is installed and Java 1.6 JVM is used. We used the default Cloudera DFS block size (i.e., 128 MB), so the data is partitioned into 128 MB blocks and a mapper is assigned to each block. As the system is assumed to have no a priori knowledge about the size of the queries, no additional partition is applied before assigning data blocks to mappers. Note that, the data and the query sets are stored in different data blocks in the DFS.

The outputs of the mappers are sent to the reducer directly without any additional combiner function. Note that, query elements labeled with "Q" are compared with data elements labeled with "D". As the types of the outputs of query and data mappers are different, aggregate calculations using combiner operation after mapping are not possible.

In our experiments, we used primarily the Enron data set [34] which is a data set of real e-mail files. The data set consists of $510,000$ e-mail files, which is equivalent to a total of $2,686,224$ KB data. The size of the files are ranged from 4 KB to 2 MB. First, the data is cleaned from the mail headers and stop words. The terms are stemmed to their roots using the *snowball stemmer* included in the Lucene [35], which is an open source library for information retrieval. We, then, calculated the *tf-idf* values of each term in each file. Finally, we created the sparse vectors of *tf-idf* values, which represent the documents. For the Enron data set, the resulted sparse vectors have $30,000$ dimensions (i.e., terms) with an average density of $160/30,000$. The density is defined as the ratio of the number of terms with non-zero

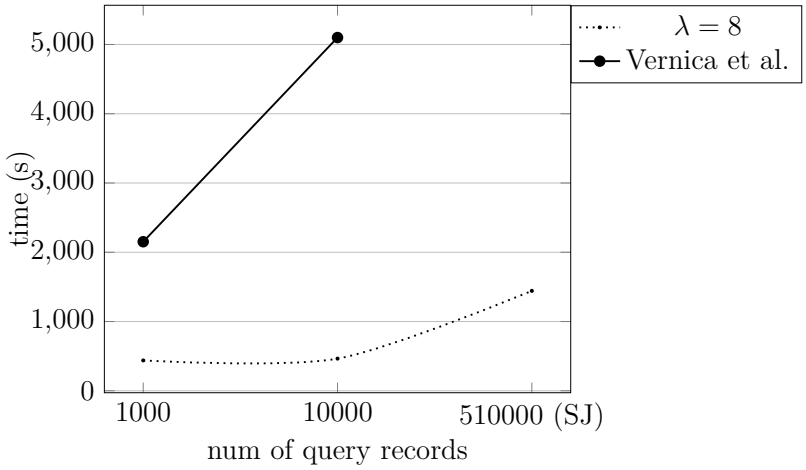


Figure 4.4: Performance Comparison between the Proposed Algorithm (ZOLIP) and the Method by Vernica et al. [1] for $k = 10$.

scores over the total number of terms in the data set [14].

Additionally, to observe the performance of the algorithm on a different data set, we used the Reuters data set [36], which consists of $20,000$ news records of 28 MB, with a dictionary size of 136 terms.

4.3.2 Performance Analysis

The performance of our algorithm is compared with the performance of the algorithm proposed by Vernica et al. [1]. Figure 4.4 shows the absolute running times of our implementations of the two algorithms for various query set sizes on the Enron data set for top 10 results (i.e., $k = 10$). As the size of the Enron data set is very large, which is overwhelming for a small three-node Hadoop cluster used in the experiments, the Java heap memory suffers from the huge amount of joins for the algorithm in [1]. Hence, the timing results for the self join case for Vernica's method is omitted in our work. A possible solution for this problem is to use a larger cluster with more nodes.

We conducted different experiments to observe the effects of different

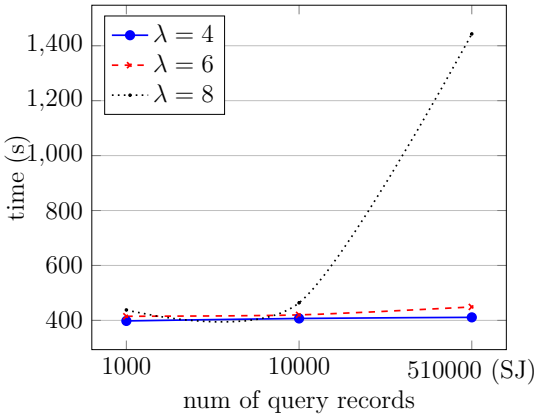


Figure 4.5: Effect of Increase in λ on Efficiency for $k = 10$

parameter settings on the performance of the proposed method. The performance of the method for different values of λ is illustrated in Figure 4.5. Figure 4.5 demonstrates that for small values of λ the effect of query size is very low. However, for $\lambda \geq 8$ search time drastically increases as query size increases, which is due to the significant increase in the number of join operations (i.e., similarity calculation between a pair of documents). A trivial remedy to decrease the computation time is to utilize a stronger cluster with more nodes.

Figure 4.6 shows the running times of each phase, namely NDD, CIT and JP, for different query set sizes. We observed that, the running time of the CIT phase increases with the number of queried documents which is expected as this phase is the main part of the algorithm where all the documents that share a common important term is compared. The NDD phase is constant with the number of queried documents. Although the complexity of this phase is linear with respect to the dictionary size (δ), it performs constant as we use a static data set. We also analyze the effect of the increase in k on the efficiency of the method and illustrate the results in Figure 4.7. As the figure illustrates, k does not have a major effect on the similarity

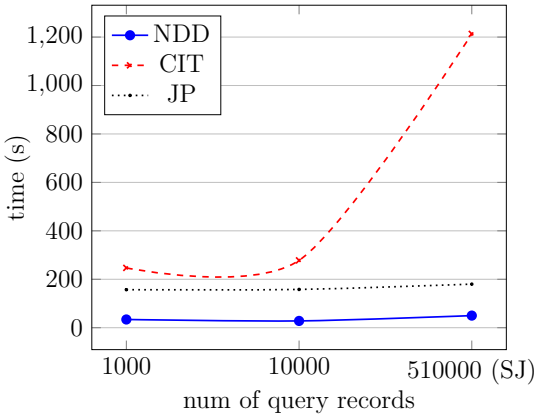


Figure 4.6: Running Time of Each Phase for $\lambda = 8$.

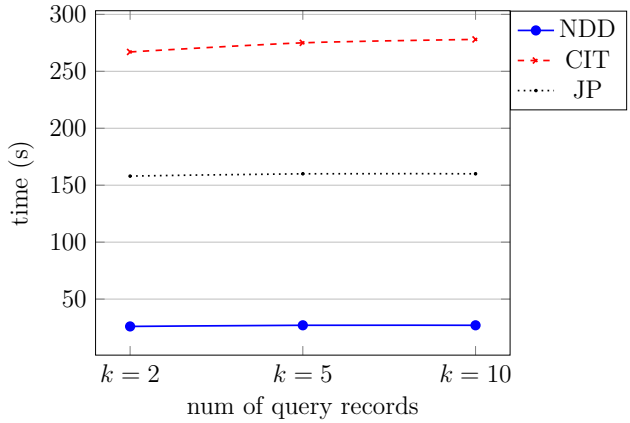


Figure 4.7: Running Time of Each Phase for different k where, Query Size is $10,000$ and $\lambda = 8$

search time. The reason for that, independent of k, given a query document our algorithm calculates the similarity scores with all other documents that share an important term with the query document and then selects the top k results. However, if the first phase (NDD) succeeds in finding k or more results then the second phase is not executed; hence the algorithm may perform better for small values of k.

Generally, from our experiments we observed that the similarity of the documents that are returned from the first phase (i.e., near duplicate detec-

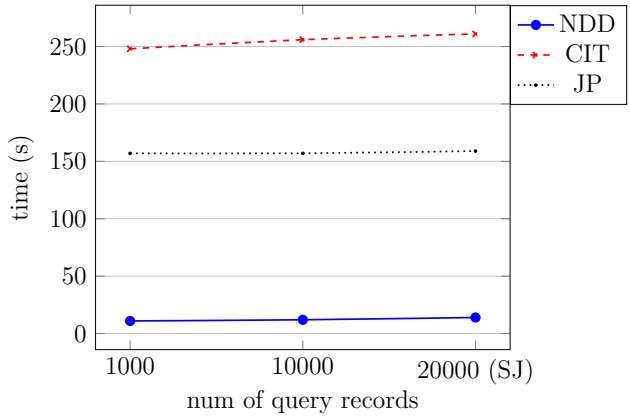


Figure 4.8: Running Times for the Reuters data set for $\lambda = 8$.

tion phase) with the query documents (i.e., $\mathcal{S}_c(q, d_i)$) are over 80 %. However, this similarity depends on the nature of the data set used as well as the parameter setting of λ.

The Enron data set is composed of e-mails, which tend to contain duplicate parts, e.g., e-mails written using a common template, which may be advantageous for the first phase (NDD) of our algorithm. Therefore, we also analyzed the performance of our algorithm using the RCV1 (Reuters Corpus Volume 1) data set, which is a corpus of news wire stories that is made available by Reuters, Ltd. [36]. This data set has about $20,000$ data elements which is much smaller than the Enron data set. The running times of each phase are demonstrated for the Reuters data set in Figure 4.8.

From Figure 4.8, it can be observed that although the Reuters data set is much smaller than the Enron data set, the search times are almost equivalent (*cf.* the running time for the Enron data set in Figure 4.6). This is due to the fact that, there are only a few near duplicate documents in the Reuters data set. While in the Enron data set, 12% (120 out of 1000 queries) of the search operations can be addressed using only the NDD phase using $k = 2$, this value reduces to 1.1% (11 out of 1000 queries) in the Reuters data set.

4.3.3 Accuracy Analysis

In this section, we analyze the accuracy of our algorithm and the trade-off between performance and accuracy. We define the accuracy of our method using the so-called *ground truth*, where the k-most similar document output of the proposed scheme is compared with the actual k-most documents calculated using the full tf-idf scores in the data set. The ratio of the cardinality of the intersection of both sets to k is defined as the accuracy rate of the algorithm. The corresponding accuracy rate of Vernica's [1] algorithm is 1.0 as they used the *PPJoin* and *PPJoinn+* [11], which is an exact similarity searching algorithm.

The parameter λ determines the accuracy rate of the proposed method. Let e_{max} be the number of bits required to represent the maximum tf-idf value in the data set and in the query set. In the proposed method during filtering, each tf-idf value is represented by only the most significant λ bits. Therefore, the tf-idf values that are less than $2^{e_{min}}$, where $e_{min} = e_{max} - \lambda$, are considered as zero in the ZOLIP filtering algorithm.

Generally, we have two types of errors that lead to loss of accuracy. In the first type, all the terms of a document have tf-idf values less than $2^{e_{min}}$. Since the terms are all filtered out, no similar document can be found for the documents with very low tf-idf values. In the second type, as the terms can only be partially represented after the filtering, there can be inaccuracies in the returned k-similar matches.

Let $\mathcal{N}_{ge}$ be the number of documents in the query set that have at least one non-zero tf-idf value in the λ-th iteration of Z-order (i.e., having at least one term with tf-idf value greater than $2^{e_{min}} - 1$). As those documents have at least one important term that is represented after the filtering, they are free of the errors of the first type. Therefore, we can formulate the rate of

the documents, for which no similar documents can be found (i.e., type 1 error) as:

$$Err_{Type_1} = \frac{|Q| - \mathcal{N}_{ge}}{|Q|}, \tag{4.1}$$

where $|Q|$ is the number of documents in the query. Note that, as λ increases, more documents are represented in the λ-th iteration on Z-order, which increases the accuracy.

In the second type of error, the results are not exactly the same as the actual k-similar documents in the ground truth. There are again two possible causes for those errors. In the first case, the first phase (NDD) outputs more than k documents. The matches found are, indeed, highly similar to the queried document as they are detected in the *Near Duplicate Detection* phase. However, the ordering can be different than the actual one in the ground truth and hence, we will miss some of the actual documents among the topmost k documents. Note that, such an error does not occur if NDD phase outputs less than or equal to k results.

The second case are related to the type-1 errors. There may be some documents in the data set that should actually be in the k similar list of a query, but missed in our method due to their low tf-idf scores. As the proposed method is a filtering technique, the final result can only be an approximation to the actual result given in the ground truth.

We tested the accuracy of the method using queries of $10,000$ documents compared with the Enron data set of $510,000$ documents. Table 4.1 shows the average number of missed queries (i.e., type-1 error) for the cases of $\lambda = 4, 6$ and 8. Note that, the error rate significantly decreases as λ increases.

Table 4.2 shows the average accuracy of the method for the queries that returns *a* result, which are the types of queries, for which type-1 error does not occur.

λ	4	6	8
Missed Queries	4309	2466	895
Average of Missed Queries	43%	25%	9%

Table 4.1: Average of missed queries of ZOLIP Filtering Algorithm with $k = 2$.

λ	4	6	8
# Queries with Inaccurate Top k	11	144	327
Accuracy	99%	98%	96%

Table 4.2: Accuracy of the top k documents for ZOLIP Filtering Algorithm with $k = 2$.

We noticed that, none of the documents for which no similar documents is found, has a term that contains a tf-idf value that appears in the λ-th iteration Z-order. Therefore, the results confirm the formula in Equation (4.1) for average type-1 error rate.

The same accuracy analysis are also performed for the Reuters data set for $\lambda = 4, 6$ and 8 with k=10, and $10,000$ documents, where the results are provided in Table 4.3.

λ	4	6	8
Average of Missed Queries	13%	12.5%	11%
Accuracy	99%	98%	96%

Table 4.3: Accuracy of the top-k documents for ZOLIP Filtering Algorithm with $k = 2$.

The average type-1 error rate can be decreased by increasing the value of $\mathcal{N}_{ge}$, which can be achieved by increasing λ. However, the increase in λ, deteriorates the performance of the method as documents that do not contain important terms would also be compared with the queried documents.

In order to decrease type-1 error rate for the query documents that do

not have any representation in the λ-th iteration Z-order, we boost the term with the highest tf-idf value so that document can now be represented in the λ-th iteration Z-order. A similar approach is adopted also for the second case of type-2 errors by boosting the tf-idf values of documents that cannot be represented.

Let B be the boosting factor and let $t_{i,max}$ be the maximum tf-idf value of document i. Then, B is defined as $B = 2^{e_{min}}/t_{i,max}$. Note that, only the term with the maximum tf-idf value is boosted and all the other values remain the same. We tested the effectiveness of the proposed boosting method using $\lambda = 8$. While the original method has 895 false matches over $10,000$ queries, the boosted method has 891 false matches which provides only a slight improvement in the number of missed queries. However, the search time is almost doubled. The reason for this degradation in performance is that, the documents that do not initially appear in the λ levels, (i.e., all the tf-idf values are very small) appear in the boosted version, which significantly increases the required number of comparisons.

Generally, in any data set, it is difficult to find the similarity for the documents that do not contain keywords with sufficiently high tf-idf values by using the cosine similarity techniques. For such a document, the Jaccard similarity metric may perform better than the cosine similarity based approaches. The proposed ZOLIP method provides efficient and very accurate results especially in data sets that contain longer documents with several important terms.

We also consider the effect of the increase in k on the accuracy of the method. We applied the ZOLIP filtering algorithm with $\lambda = 8$ and we set $k = 10$ to check if it reduces the accuracy of our filtering algorithm. The increase in k does not affect the accuracy in any significant manner, as shown

in Table 4.4.

k	2	5	10
wrong k-Similar	327	384	433
Accuracy	96%	95.6%	95.2%

Table 4.4: Accuracy of ZOLIP Filtering Algorithm with Different Values of k when $\lambda = 8$.

We also measured the accuracy of the method using Relative Error on the Sum of distances(RES) [37] metric. Let $S_{q,k}$, be the top-k similar result from a query q. And let also $G_{q,k}$ be the ground truth of the top-k similar documents, then RES is defined as

$$RES(q, S_{q,k}) = \frac{\sum\limits_{x \in S_{q,k}} \mathcal{S}_c(q, x)}{\sum\limits_{y \in G_{q,k}} \mathcal{S}_c(q, y)} - 1.$$

RES is calculated for the Enron data set with $10,000$ queries, using $\lambda = 8$, and $k = 2, 5, 10$. Table 4.5 shows the relative error values after removing the queries with type-1 error from the calculations.

k	2	5	10
RES	0.0262	0.0216	0.0178

Table 4.5: Relative Error on the Sum (RES) for Different Values of k when $\lambda = 8$.

4.4 Conclusion

In this chapter, we propose an efficient filtering method based on the Z-order prefix that can be used in document similarity algorithms with cosine similarity metric. The algorithm provides effective and efficient results. In

particular, it is very fast in returning highly similar documents. We demonstrate that our method performs much better than the algorithm in [1] for the cases, where the density of the vectors is relatively high. This is especially true when the variance in the tf-idf values are high. This is a general observation in data sets, in which majority of the documents can be represented accurately with only several terms of relatively high tf-idf values.

On the other hand, our algorithm have some limitations for documents, all of whose terms have very low tf-idf values. The documents with terms of very low tf-idf values, however, imply that they cannot be accurately represented by a particular subset of terms. Therefore, a filtering based on important terms do not produce accurate results for those kinds of documents. Instead, we propose to use a hybrid filtering method based on the Jaccard similarity metric for documents with no important terms while our filtering technique can be used for the documents with at least several important terms.

The proposed method provides results with very high accuracy if the accuracy parameter λ is set to a large value. However, for relatively small values of λ, the search time significantly decreases while still satisfying reasonable accuracy. There is a trade off between accuracy and efficiency, hence the accuracy parameter λ should be set according to the requirements of the application and the data set. For instance, in finding highly similar documents in applications such as duplicate web site and plagiarism detection, a relatively small values of λ can produce results with high accuracy.

Chapter 5

SECURE DOCUMENT SIMILARITY SEARCH UTILIZING SECURE SKETCHES

This work considers the problem of detecting the identifiers of the most similar documents within an encrypted data set for a given query document, without revealing the features (i.e. terms) of neither the data set elements nor the query document [38]. In order to apply secure search, a metadata called *sketch* is used. The sketches can hide the feature information of the underlying data but still preserve similarity properties. Prior to outsourcing, a sketch per data set entry is generated by the data owner, and the search is applied over the sketches without using the encrypted data itself.

We provide two approaches with different privacy guarantees. The first one uses the sketches as is and provide very efficient search capability. While this method can perfectly hide the features of both documents and the query,

search and access patterns are allowed to leak. Although these patterns are also allowed to leak in the majority of the other works in the literature, they may leak considerable sensitive information especially in the existence of some background knowledge [39]. Hence, we also propose an approach with enhanced security properties such that both search and access patterns are hidden in addition to the document and query features. The approach with enhanced security requires encrypted sketches, which utilizes costly homomorphic encryption techniques.

The rest of the work is organized as follows. Problem definition is given in Section 5.1. An introduction to our algorithm and the approaches used in this chapter is described in Section 5.2. Our secure sketch constructions are provided in Section 5.3. Enhanced security search described in Section 5.4. Security analysis are given in Section 5.5 Section 5.6 provides the implementation details of the algorithm utilizing map reduce computation model over the Hadoop distributed file system (HDFS) and Section 5.7 shows the experimental results. Finally, Section 5.8 concludes the work.

5.1 Problem Definition

In this section, we formalize the secure similarity search problem and provide the definitions and tools that are used throughout the chapter. The common notations that are used extensively in this chapter are summarized in Table 5.1.

Definition 9 (Secure k-Similarity Search (SSS)) *Let D be the outsourced data set and $d_1, \ldots, d_n$ be the n records in D and $Q =< q_1, \ldots, q_m >$ be a query with m features (i.e., attributes). Further let E_{sk} be an encryption method with key sk. Then, secure similarity search (SSS) protocol is defined*

Table 5.1: Common Notations

D	data set.
Q	query set.
d_i	i^{th} document in the data set.
q_i	i^{th} query document in the query set.
$Sk(d_i)$	secure sketch for document d_i
λ	size of the sketch.
δ	number of terms in the data set.
v_i	random vectors of $\{-1, 1\}$ with δ elements.
d_{cos}, d_{ham}	cosine and hamming distances.
$\oplus_S$	secure *xor* operation

as:

$$SSS(E_{sk}(D), E_{sk}(Q)) \rightarrow < d'_1, \ldots, d'_k >,$$

where d'_i is the identifier of the i^{th} nearest record to Q in D.

Definition 10 (Data Confidentiality) *Given a k-similar document protocol, let D be the data set outsourced to the cloud. An SSS protocol provides data confidentiality if the contents of the documents $d_i \in D$ are not revealed to the cloud server.*

Definition 11 (Query Privacy) *Given an SSS protocol, let Q be a query that its similar documents are searched for. An SSS protocol provides query privacy if the features of Q are not revealed to the cloud server.*

Definition 12 (Similarity Pattern Privacy) *Given an SSS protocol, let Q and Q' be two queries that their similar documents are searched for. An SSS protocol provides similarity pattern privacy if it is not possible for an adversary, including the cloud, to detect whether the queries contain a subset of common features or not.*

Definition 13 (Access Pattern Privacy) *Given an SSS protocol, let Q be a query that its similar documents are searched for. An SSS protocol provides access pattern privacy if access records corresponding to the k most similar documents of Q is not revealed to any party other than the owner of the query.*

5.2 Secure Similarity Search

In this section, we present our secure search scheme based on locality sensitive hash (LSH) functions. We consider the data outsourcing scenario, where the data is outsourced to an honest but curious cloud server. As the data may contain sensitive information, a secure metadata called sketch is generated using the original data and only the metadata (i.e., secure sketches) is shared with the cloud. We construct two different search schemes. The first one shares the sketches with the cloud in the plain form hence all the calculations can be performed very efficiently. But this scheme may leak some valuable information such as the access pattern. The second approach shares encrypted sketches. While this scheme provides very high level of privacy, the computation cost is significantly higher due to bit-wise homomorphic encryption operations.

In the first scheme, there are three entities namely: data owner, multiple users and an honest but curious cloud server. In the case of the second scheme that uses encrypted sketches, in addition to those three entities there is also a second honest but curious server that we call proxy. Initially, the data owner encrypts the data and creates searchable sketches and then, outsources the sketches to the cloud. Given a query, the cloud server calculates the list of similar documents (with the help of the proxy in the second approach) and

returns a list of the identifiers of the most similar documents.

5.3 Secure Sketch Construction

The aim of using sketches is to represent documents by small, constant length binary vectors. While the sketches obfuscate the content (i.e., features) of the documents, it is still possible to estimate the similarity of the underlying documents from the sketches only [29]. The sketches are generated using the principle of *locality sensitive hashing* (LSH). In LSH functions, different from the cryptographic hash functions, similar inputs provide the same output with high probability and dissimilar items provide different outputs with high probability.

Our method is based on the cosine distance between document pairs. We represent each document feature set as a point on spaces that have multiple dimensions such that each feature is represented by an axis and the tf-idf score of that feature is its value in that axis. The cosine distance between two points is the angle that the vectors to those points make. Note that, independent from the number of dimensions (i.e., features), the cosine distance will be in the range 0 to 180.

Suppose we have two vectors x and y and pick a random hyperplane through the origin. The hyperplane intersects the plane of x and y in a line. Consider a vector v that is normal to the hyperplane. Either x and y will be on different sides, which means vx and vy have different signs, or they will be on the same side, meaning vx and vy have the same sign. Since the hyperplane is chosen randomly, with $\theta/180$ probability vx and vy will have different signs, where θ is the angle between x and y. Note that, the higher the similarity of two vectors (i.e., smaller θ), the higher the probability that

they have the same sign, which is in parallel with the principle of LSH.

Let the size of the feature list which is the number of all the possible keywords in the data set, be δ. In the construction of the sketches, first a collection of λ vectors, $(v_1, \ldots, v_\lambda)$, with δ dimensions are chosen, where the elements of the vectors are randomly chosen as either $+1$ or -1. Given a document vector d_i, the inner product of d_i with each vector v_j is calculated and the elements of the sketch of d_i are set according to the signs of those inner products. The sketch of d_i, which is a binary vector that is denoted as $Sk(d_i) = \{s_{i1}, \ldots, s_{i\lambda}\}$, is constructed as follows:

$$Sk(d_i) = \{s_{i1}, \ldots, s_{i\lambda}\},$$
$$s_{ij} = \begin{cases} 1 & \text{if } d_i v_j \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Note that, $\forall k \in \{1, \ldots, \lambda\}$ and $\forall d_i, d_j \in D$,

$$prob\left(Sk(d_i)[k] = Sk(d_j)[k]\right) = 1 - \frac{d_{cos}(d_i, d_j)}{180}.$$

This implies that, if $d_i = d_j$ then $Sk(d_i) = Sk(d_j)$ and the number of common sketch elements gets higher as the cosine distance between the corresponding documents gets smaller.

Then, we formally define the similarity (S) of two documents d_i, d_j as follows:

$$S(d_i, d_j) = \lambda - d_{ham}(Sk(d_i), Sk(d_j)), \tag{5.1}$$

where d_{ham} denotes the hamming distance.

This approach successfully hides the information of features and their corresponding scores. However, the generation of sketches is a determinis-

tic operation. Therefore, the server may identify if a document is queried multiple times or may learn the identifiers of the documents in a query's similar document list. Those are the similarity pattern and access pattern information respectively. This approach is very efficient and can especially be used in scenarios, where the response time is crucial but the leakage of search and access patterns may be acceptable. Indeed, most of the works in the literature of secure search, allows the leakage of these two patterns for efficiency concerns.

The next section explains the construction of encrypted sketches, which also hides the search and access patterns but performs slower due to the complex bit-wise homomorphic encryption operations.

5.4 Enhanced Security

The secure sketches obfuscates the information of the features in the documents since each bit of a sketch in calculated by multiplying the document feature vector with a random vector and then reducing the result to a single bit by just using its sign. However, the sketch construction is a deterministic operation. Therefore, the same collection of λ random vectors is used in the construction of the sketches for all documents in the data set. Deterministic construction reveals the similarity pattern, namely server can learn if the same feature vector is queried before. Moreover, since the sketches are stored in plain in the server, the server can learn the identifiers of the documents that are similar with the queried document, which is the access pattern.

In order to hide both search and access patterns, we encrypt the sketches. However, since the server should apply a similarity search operation, an homomorphic encryption method is used. Homomorphic encryption methods

can perform some operations over the encrypted data without knowing the private key. We use the Paillier encryption [40] as the underlying homomorphic encryption method. It provides two homomorphic properties, addition and constant multiplication on field Z_N as:

$$E(a+b) = E(a) \cdot E(b) \mod N^2$$
$$E(ab) = E(a)^b \mod N^2$$

As shown in Equation 5.1, the similarity between two sketches is calculated using the hamming distance.

Hence, in the case of encrypted sketches, the server calculates the encrypted similarity using the encrypted hamming distance between the two encrypted sketches. Note that, hamming distance is just the summation of the bits of the xor of two binary input vectors. A secure xor operation that works over encrypted input vectors is explained as follows.

Secure XOR ($\oplus_S$)

Note that, XOR operation can be calculated by addition and multiplication operation as:

$$a \oplus b = a + b - 2(ab).$$

Hence a secure xor on modulo N, denoted with $\oplus_S$, can be calculated as:

$$\begin{aligned} E(a) \oplus_S E(b) &= E(a \oplus b) \\ &= E(a) \cdot E(b) \cdot E(ab)^{N-2}. \end{aligned} \tag{5.2}$$

While Paillier encryption provides homomorphic addition, homomorphic multiplication of two ciphers is not trivial. Homomorphic multiplication with Paillier requires a second party, that has the decryption key. Although, this

second server knows the Paillier private key, it does not need to be trusted as long as the two servers not collude. Algorithm 4 shows the secure multiplication method. The server randomizes the two multiplication parameters by homomorphically adding a random number and sends the randomized parameters to the second server. The second server (i.e., proxy) decrypts and multiplies the given two values. Then, it encrypts the results and sends them back to the first server. Finally, the first server homomorphically subtracts (i.e., multiply with modular inverse) the blinding factor of the random values and gets the encrypted multiplication result.

Algorithm 4 Secure Multiplication ($E(ab)$)

procedure P1(a,b)
  pick random $r_1, r_2 \mod N$
  $a' = E(a) \times E(r_1) = E(a + r_1)$
  $b' = E(b) \times E(r_2) = E(b + r_2)$
end procedure
procedure P2(a', b')
  $D(a') = a + r_1$
  $D(b') = b + r_2$
  $h = D(a') \times D(b') \mod N$
  $\triangleright\ h = ab + br_1 + ar_2 + r_1 r_2 \mod N$
  $h' = E(h)$
  send h' to P1
end procedure
procedure P1(h')
  $E(ab) = h' \times E(a)^{N-r_2} \times E(b)^{N-r_1} \times E(r_1 r_2)^{N-1} \mod N^2$
end procedure

The cosine similarity value can be calculated by using secure xor and homomorphic addition operations over the sketches. Each element (i.e., bit) of the sketches are first encrypted with the Paillier encryption method, using the public key k_p as $E(Sk(d_i)) = \{E(Sk(d_i)[1]), \ldots, E(Sk(d_i)[\lambda])\}$. The hamming distance between two encrypted sketches is calculated by first calculating xor of each sketch bit and then applying summation over the results,

as shown in equation (5.3).

$$E(d_{ham}(Sk(d_i), Sk(d_j))) = \\ = \sum_{k=1}^{\lambda} E(Sk(d_i)[k]) \oplus_S E(Sk(d_j)[k]). \quad (5.3)$$

Given the encrypted document sketches $\mathcal{S}k = \{E(Sk(d_1)), \ldots, E(Sk(d_{|D|}))\}$ and a query sketch $E(Sk(q))$, the server finds the encrypted hamming distance between the query and each other document. Calculating hamming distance requires calculating secure xor ($\oplus_S$) between encrypted sketches, which is performed with the participation of the proxy. The encrypted scores are then shuffled by a random permutation given by the user. This shuffling hides the correlation of the scores with the actual identifiers from the proxy. Since the users choose a random permutation at each query, it is not possible for the proxy to apply any kind of inference attack. The list of shuffled encrypted scores is then sent to the proxy, which decrypts and sends to the user. The user applies the permutation back and learns the actual document identifiers with the minimum hamming distance (i.e., maximum similarity). The secure similarity search method is described in Algorithm 5.

5.5 Security Analysis

In this section, we provide the formal definitions and proofs that the proposed scheme is secure.

In the proposed method, both the document and the query sketches are generated in an identical procedure, hence the definitions data confidentiality and query privacy are equivalent for this method.

The secure sketch method (without encryption), provides data confiden-

Algorithm 5 Enhanced Secure Similarity Search

SERVER:

Require: $\mathcal{S}k$: encrypted document sketches

$E(Sk(q))$: query sketch, p: random permutation

for all $E(Sk(d_i)) \in \mathcal{S}k$ **do**

$d_{ham} = 0$

for $j = 1 \rightarrow \lambda$ **do**

$d_{ham}[j] += E(Sk(d_i)[j]) \oplus_S E(Sk(q)[j]$

end for

$S(q, d_i) = d_{ham}$

end for

shuffle encrypted scores $S(q, d_i) \in \mathcal{S}$ with random permutation p

send shuffled encrypted scores $\mathcal{S}$ to proxy

PROXY:

Require: $\mathcal{S}$: shuffled encrypted scores

Decrypt each score in $\mathcal{S}$

send scores to user

USER:

apply permutation p on the scores to map with real identifiers

sort scores to find identifiers with min hamming distance

tiality. Each sketch element s_{ij} is just the sign (positive or negative) of the inner product of document feature list d_i, with a random vector v_j. The random vectors are secret information, hence not revealed to the server. The only information leaked from a query or document sketch is the information of $d_i v_j \geq 0$ or not, where v_j is random and not known by the server. Therefore, s_{ij} is also random and does not reveal any information about the content of d_i.

This method somehow leaks the similarity pattern. Two sketches that are generated with the same feature lists (i.e., $d_i = d_j$) will be the same due to the deterministic sketch generation operation. However, it is important to note that, identical sketches do not necessarily imply identical feature lists.

$$d_i = d_j \implies Sk(d_i) = Sk(d_j), \text{but}$$
$$Sk(d_i) = Sk(d_j) \not\Longrightarrow d_i = d_j.$$

But if two sketches have high number of common elements, then the probability that, the two underlying feature lists also have common elements, is high.

Similarly, the access pattern is also leaked. Since the server can calculate the similarity score between the query and the documents, the information of the k most similar documents of the query is revealed.

The enhanced method proposed in Section 5.4 provides both similarity and access pattern privacy in addition to the data confidentiality and query privacy. Note that, in the method with encrypted sketches, each element is encrypted with a semantically secure encryption method, which means each encrypted sketch bit is indistinguishable from a random number. Therefore, it is not possible to distinguish queries with identical feature sets, hence

similarity pattern is protected.

In the encrypted sketch method, the server works only on the encrypted values so cannot learn anything about the scores. Therefore, access pattern is not leaked to the server. The proxy has the decryption key and calculates the scores, but in each query, the server shuffles the encrypted score list with a random permutation given by the user before sending to the proxy. Since users pick a different random permutation at each query, access pattern is also not revealed to the proxy.

5.6 Implementation

We utilize the MapReduce computing model and the Hadoop framework in our implementation. We developed two MapReduce phases in our approach. While the first phase calculates the secure xor operation between the query sketch and data set sketches, the second phase sums up the results to calculate the hamming distance.

There are two mappers in the first phase: one for the query and one for the data set elements. Both mappers read the sketches and generate key - value pairs by assigning an index to each encrypted sketch bit as key and encrypted bit as value. The only difference between mappers is that, query and data set elements are distinguished by a prefix "Q:" and "D:" respectively. The output of the mappers is in the format $< index, D/Q : d_j, E(S_j[p]) >$, where p is the index of the encrypted sketch bit of document ζ and D/Q is the prefix that represents query or data set element. The reducer of the first phase gets the output of the mappers and combines the values (i.e., encrypted bits) that have the same key. Then, it calculates the secure xor operation as explained in equation (5.2). The result of this operation returns the encrypted xor of

two encrypted sketch bits, in the format;

$$<< q, d_j >, E(S_q[p] \oplus S_j[p]) > .$$

The second phase calculates the encrypted hamming distance between the sketches using the output of the first phase. This phase has only a reducer without a map function. For each document in the data set, the reducer collects the keys that have the same query-document pair (i.e having the same q, d_j) and then homomorphically calculates the sum of all the encrypted xor outputs. The result of the summation is the encrypted hamming distance. Note that, the smaller the distance, the higher the cosine similarity.

5.7 Similarity Evaluation

In this section, we evaluated the accuracy and computation costs of the secure similarity search methods. The Paillier cryptosystem is used and the proposed methods are implemented in Java language. The experiments are conducted on Cloudera CDH4 with a cluster of 3 nodes. Two nodes with Xeon processor E5-1650 3.5 GHz with 12 cores 15.6 GB of RAM and 1 TB hard disk and one node with Core i7 3.07 GHz with 8 cores, 15.7 GB of RAM and 0.5 TB hard disk. On each node, Ubuntu 12.04 LTS, 64 -bit operating system is installed and Java 1.6 JVM is used.

The Enron data set [34] is used to evaluate our system which is a collection of $510,000$ real e-mail files. The size each file changes from 4 KB to 2 MB. First, the data is cleaned from the mail headers and stop words. The terms are stemmed to their roots using the snowball stemmer included in the Lucene [35] library. We then, calculated the *tf-idf* values of each term in each file and created the sparse vectors that represent the documents and the

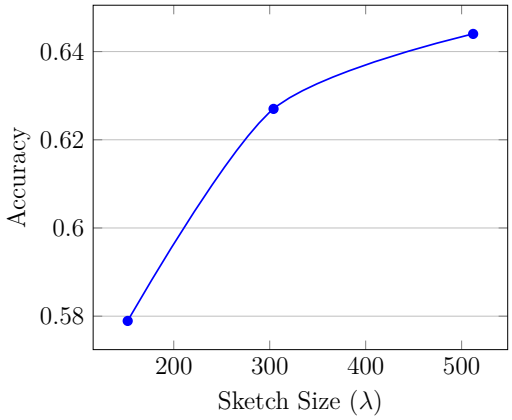


Figure 5.1: Average Accuracy Rate

corresponding *tf-idf* values. Finally the sketches are generated using these sparse vectors and outsourced to the cloud.

The sketching method used in our protocol provides an estimation for the similarity. The accuracy of the method is measured by comparing the size of the intersection set of the top-k results returns from our protocol and the actual top-k results for different values of k that ranges from 5 to 20. Note that, the accuracy of the method increases as the size of the sketches increase. However, this also immediately increases the computation costs. Fortunately, as Figure 5.1 shows, the accuracy is high even for moderate sketch sizes.

We evaluated the computation costs of both sketch search and encrypted sketch search methods. As expected, the sketch search is very efficient. The computation of k most similar documents of a query among the whole data set of size $510,000$ documents is in the order of seconds as shown in Figure 5.2.

We also evaluated the computation costs for the enhanced security case. In this case each sketch bit is encrypted with the Paillier encryption for 512-bit key size. As can be observed from Figure 5.3, the running time grows

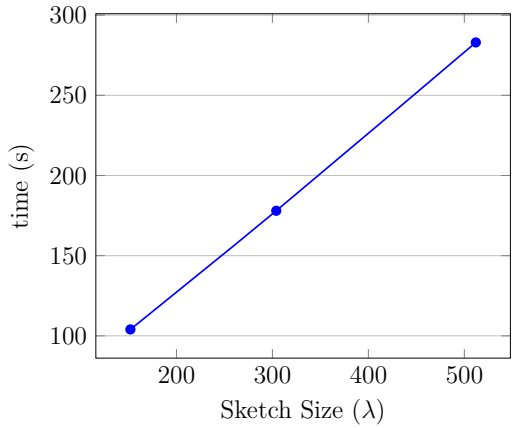


Figure 5.2: Time Complexity for Sketch Similarity Search, $|D| = 510,000$

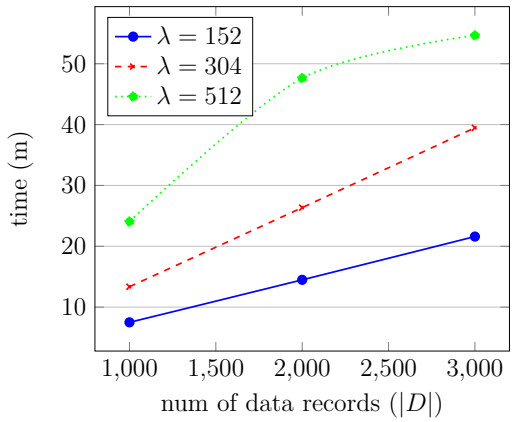


Figure 5.3: Time Complexity for Encrypted Sketch Similarity Search

linearly with sketch size and data set size. However, the requested number of similar items k, has no effect on the running time. The method sorts all the documents according to its relevancy with the query document and returns the top k results.

Most of the works in the literature consider the problem of similar document detection among two parties, where one party compares its plain data with the encrypted data of the second party. However, this problem is not suitable for the data outsourcing scenario that we consider. The only work that provides equivalent security properties with our work is the secure k-

nearest neighbor search, by Elmehdwi et al. [25]. That work also considers the data outsourcing scenario with hiding search and access pattern but for a structured data set for attributes with numeric values. They also utilize two servers similar to our approach but finds Euclidean distance instead of the cosine distance we consider. The protocol in [25] is bounded by $O(n*(l+m+k*l*\log_2 n)$ encryption and exponentiation operations, where n is the data set size, m is the number of attributes, l is the domain size (in bits) of Euclidean distance in data set and k is the number of top similar items. Our protocol does λ secure XOR operations and $\lambda - 1$ homomorphic addition operations per each data record comparison. Each secure XOR has 1 exponentiation and 2 multiplication operations (i.e., constant) hence, the protocol is bounded by $O(n*\lambda)$ where, λ is constant. Since our protocol is independent from k and the number of features, it outperforms [25], especially for large values of k.

5.8 Conclusion

In this chapter, we address the secure similar document detection problem for cosine similarity, using locality sensitive hash functions. Our approach is based on creating a fixed length sketch per each document. We considered two different security levels. The basic method uses plain sketches and provides a very efficient implementation. The other method enjoys an enhanced security protection by using encrypted sketches, where each sketch element is encrypted with Paillier encryption method. The security analysis of both methods are provided and the efficiency and effectiveness of the method is demonstrated through extensive set of experiments. Generally,the First level shows some applicability for using the algorithm in the mean time, while the

other secure algorithm shows good theoretical accuracy, unfortunately it still inapplicable in the present.

Chapter 6

SECURE DOCUMENT SIMILARITY SEARCH UTILIZING MINHASH

In this chapter, we adapt an existing secure searchable indexing method to k-nearest neighbor (k-NN) for documents. Throughout the chapter we refer the method also as a method for finding top-k similar documents or document similarity search. In the method, the document similarity search is handled in a privacy-preserving manner. The method uses locality sensitive hashing (LSH) together with homomorphic encryption to construct a searchable index that allows secure search operations while the actual data itself is protected using symmetric encryption against an adversary including especially the cloud storage. More specifically, the proposed method provides data and query confidentiality.

The method is implemented using MapReduce parallel programming framework on a Hadoop cluster of three computers. The implementation results show that it can effectively be used in moderate sized data sets and it is

scalable for much larger data sets provided that the number of computers in the Hadoop cluster is increased.

6.1 The Framework

We consider a data outsourcing scenario that consists of three entities: data owner, two non-colluding semi-honest servers and query users. The big picture for the interactions between the entities is illustrated in Figure 6.1.

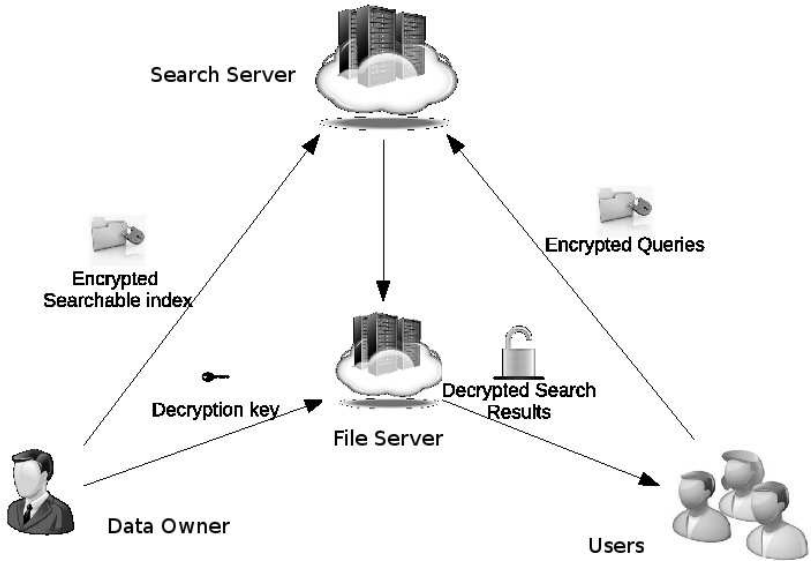


Figure 6.1: The framework

Data Owner is a person or organization that owns a data set and wants to outsource the database functionalities along with the data to the cloud. Since the data may contain sensitive information, it is encrypted prior to its outsourcing to the cloud. Beside the encrypted data, a searchable index is also generated by the data owner for enabling secure search operations and shared with the cloud. It is important to note that, searchable index and encrypted data are outsourced to two different non-colluding servers as explained subsequently.

Cloud Server is a party that offers storage and computational services. We assume the cloud servers are semi-honest (i.e., honest but curious), a common model for cloud environment. In our settings we utilize two non-colluding servers: *file server* and *search server*.

The search server takes a query from a user, applies the requested search operation over the secure index and returns encrypted intermediate results to the file server. Then, the file server decrypts and sorts them and send the final results to the user. The aim of using two non-colluding servers is to hide the correlation between the queries and final results. The search server learns the queries, but not what they match with. Similarly, the file server learns the requested data but not the query itself.

Users are the authorized entities that have the right to query the cloud data. A user employs certain cryptographic techniques to protect the query confidentiality. A query generated in this manner is then sent to the search server. The user receives the search results from the file server.

We can formalize the secure operations over the encrypted data as follows. Let $\mathcal{D}$ be a database with n data records and $\mathcal{F}$ be the set of all features (terms or words), which are included in data entries or queries. There are five main phases in the method, namely: *Setup*, *Trapdoor Generation*, *Index Generation*, *Query Generation* and *Search*.

1. *Setup (ψ)*: Given a security parameter (ψ), it generates a symmetric key and a public key pair as $\mathcal{K} = \{K_{id}, (K_{pub}, K_{pr})\}$.

2. *Trapdoor Generation(Δ)*: Given a list of terms Δ (i.e., dictionary), it generates a set of random permutations of Δ as $\mathcal{T} = \{P_1, \ldots, P_\lambda\}$, where λ is a precision parameter.

3. *Index Generation ($\mathcal{K}$,$\mathcal{D}$)*: Given a database ($\mathcal{D}$), a secure searchable in-

dex $\mathcal{I}$ that represents each entry in $\mathcal{D}$ is generated by using the features of the entries in $\mathcal{D}$ and the key $\mathcal{K}$.

4. *Query Generation*$(\mathcal{T}, F)$: Given the trapdoors $\mathcal{T}$ and a set of features to be queried, it generates a secure query Q for the given features that does not reveal the corresponding features.

5. *Search*$(\mathcal{I}, Q)$: The given query Q is compared with the searchable index $\mathcal{I}$ and the encrypted matching entries from $\mathcal{D}$ are returned.

6.2 Security Model

In this section, we analyze the security of the proposed scheme in passive adversary model as it is a common adversarial model considered in secure data outsourcing scenario [41]. We assume the adversary is anyone on the search server side (e.g., a database administrator) as it is the entity that has the access to the secure index and secure queries and may try to extract any sensitive information that may be leaked from them. In this model, the server is considered honest but curious; i.e., it implements all storage and processing functions correctly, but learns from any information leak.

The main privacy requirements that need to be satisfied are data and privacy confidentiality, whose intuitive definitions are given in Definitions 14 and 15. The privacy can further be extended to hide some side information such as pattern confidentiality. Violation of pattern confidentiality may cause the system be subject to some adversarial analysis [39]. Although pattern confidentiality can be provided with oblivious RAM techniques [42] [43], due to efficiency concerns, we allow to leak that information in this work.

Definition 14 (Data Confidentiality) *A secure search operation provides data confidentiality if the outsourced data (i.e., encrypted data and searchable meta-data) does not leak the information of the actual content or features of the data set elements.*

Definition 15 (Query Confidentiality) *A secure search operation provides query confidentiality if the given query does not leak the information of the actual queried terms or features.*

6.3 Proposed Method

This section provides the details of our proposed scheme for the document similarity search method. Secure search over encrypted data is possible via a secure searchable index generated by data owner. The search index is then sent to the cloud server together with the actual encrypted data. While it is still possible to search for the desired documents, secure index prevents the cloud service provider (i.e. search server) from learning sensitive information about the actual data (see Figure 6.1).

6.3.1 Secure Index Generation

In our document similarity searching algorithm, we utilize the bucketization technique developed by Kuzu et. al. [44]. Bucketization is a partitioning technique which distributes each object (e.g., documents in our case) into a constant number of buckets according to the outputs of the MinHash functions (i.e., signatures). Due to the principle of locality in MinHash functions, the probability of two documents be assigned to the same bucket is the same as their Jaccard similarity.

The main idea of our work is to create a signature based on MinHash functions to represent each document in the data set. The signatures are used to compare the corresponding documents. The proposed method for constructing the secure index consists of three phases, namely: feature extraction, bucket index construction and bucket index encryption. These phases, as illustrated in Figure 6.2 and explained in the subsequent sections, are performed the data owner.

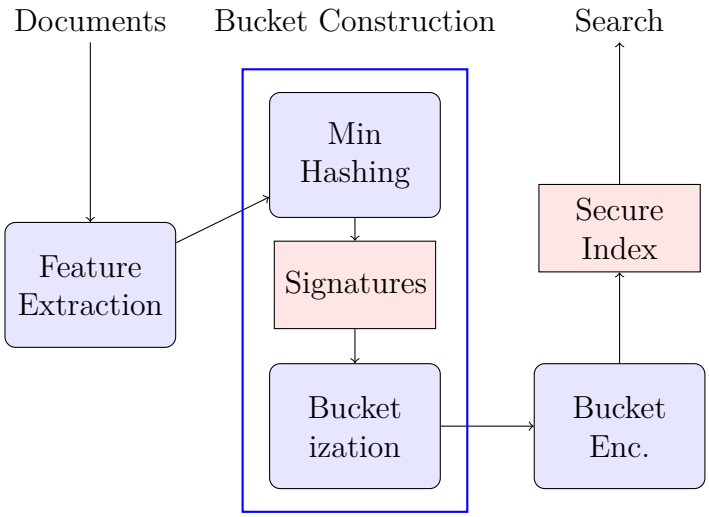


Figure 6.2: Flowchart of secure index generation

Feature Extraction

For each data element $D_i \in \mathcal{D}$ feature set $F_i = \{f_i^1, \ldots, f_i^y\}$ is extracted. In the case of documents of text data, which are used for k-NN search, each feature is composed of pairs $f_i^j = (w_{ij}, rs_{ij})$, where w_{ij} is the word j in document i, and rs_{ij} is the relevancy score of the term in the corresponding document. In this work, the tf-idf value is used for relevancy scoring. This relevancy score is then used to rank the results of the queries in the search process. Since relevancy scores (rs_{ij}) are sensitive information, they are up-

loaded to the cloud only after encryption. There is a family of cryptographic algorithms, known as homomorphic cryptosystems, that allow certain operations to be performed over the encrypted data (ciphertext) and generates an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. In order to take advantage of these homomorphic properties, we use Paillier encryption [40], which is a well known additive homomorphic encryption method. The relevancy scores are encrypted with the Paillier scheme and all the numerical calculations on the server are performed over these encrypted values thanks to its homomorphic properties.

In a feature set F_i, there can be several terms with very low relevancy scores. Representing all those terms with low importance has an adverse effect on the accuracy of the method, since they may obstruct terms with high relevancy score in the signature. Therefore, we select only the terms with tf-idf values higher than a predefined threshold σ (i.e., only the important terms are used). In the case, where there is no or only very few terms with tf-idf values higher than σ, we select t_{min} terms with the highest relevancy scores.

Bucket Index construction

In the bucket index construction there are two phases. First, using the feature sets and the MinHash functions, a constant length signature is generated for each data set entry. Each feature set is hashed with λ MinHash functions, where each function is under a random permutation P_j of the set of all possible features Δ. Let F_i be the feature list of document $D_i \in \mathcal{D}$, then the signature of D_i is calculated as

$$Sig(D_i) = \{h_{P_1}(F_i), \ldots, h_{P_\lambda}(F_i)\}. \tag{6.1}$$

The signatures are sets of λ elements. An important property of signatures is that, the similarity between signatures of two data items represents the similarity of the underlying feature sets of the corresponding data items as discussed in Section 3.4.

After the generation of signatures, each data item is distributed to λ different bucket according to their signatures. Let B_k^i be the bucket identifier for the i^{th} MinHash with output k, the content vector VB_k^i is defined as:

$$(id(D_j), rs_{jk}) \in V_{B_k^i} \textit{iff } D_j \in B_k^i. \tag{6.2}$$

Note that, independent of the number of its features, each data element is mapped to exactly λ different buckets, but the total number of different buckets depends on the set of features Δ, the feature set of each data element (F_i) and the randomly chosen MinHash functions.

Bucket Index Encryption

Both the bucket identifiers and the bucket content vectors contain some sensitive information, which need to be protected prior to outsourcing. The bucket identifier represents the MinHash function used and its output. This may reveal some important information of the input feature set such as the terms it does not contain and one term that it definitely contains. Therefore, bucket and query contents should be protected using a cryptographic primitive. However, the server also needs to be able to match the queried encrypted buckets to the buckets stored in the server. This requirement necessitates using a deterministic cryptographic scheme for protecting the bucket identifiers. HMAC functions can be considered as cryptographic hash functions that take a secret key as input besides the message. We use HMAC functions

to hide the information that can leak from bucket identifiers. As the process is deterministic comparison of buckets is still possible. The HMAC secret key (K_{id}) is not revealed to the server, hence it is secure against any brute-force attack. The encrypted bucket identifier is denoted as

$$\pi_{B_k^i} = HMAC_{K_{id}}(B_k^i). \tag{6.3}$$

On the other hand, the bucket content vector stores the document identifiers and relevancy scores of the data elements, which are mapped to that bucket. In order to get the score of a document, it is required to map the bucket elements with the same document identifier. Therefore, the document identifiers are again hashed with a cryptographic hash function that is denoted as $H(id(D_i))$. For the relevance scores, we use additive homomorphic encryption methods that permit homomorphic addition operation over the encrypted data without requiring decryption. More specifically, we use the Paillier encryption scheme [40], which is a very efficient additive homomorphic encryption technique. Formally, Paillier encryption satisfies the following fundamental homomorphic property,

$$E(m_1, r_1) \cdot E(m_2, r_2) = E(m_1 + m_2, r_3),$$

where m_1 and m_2 are two messages and r_i values are random numbers.

A public encryption key (K_{pub}) for the Paillier cryptosystem is used for encrypting each bucket element before outsourcing the data to the server. The result of the encryption bucket contents is a list of pairs as

$$\pi_{B_k^i} = \{(H(id(D_j)), E_{K_{pub}}(rs_{jk})\} \forall id(D_j) \in B_k^i. \tag{6.4}$$

Paillier encryption provides semantic security against chosen plain text attacks, which means multiple encryptions of the same message result in different ciphertexts, which cannot be linked. The secure index generation method is summarized in Algorithm 6.

Algorithm 6 Secure Index Generation

Require: Δ: set of possible keywords, $\mathcal{D}$: database, h: *MinHash* function, Ψ: security parameter
Ensure: $\mathcal{I}$: Secure index for $\mathcal{D}$

$K_{id}, K_{pub}, K_{priv} \leftarrow Setup(\Psi)$
$\mathcal{L} \leftarrow \emptyset$ ▷ $\mathcal{L}$: Bucket Identifier List
for all $D_i \in \mathcal{D}$ **do**
    $F_i \leftarrow$ extract features of D_i
    $Sig(D_i) \leftarrow \{h_{P_1}(F_i), \ldots, h_{P_\lambda}(F_i)\}$
    for $j = 1 \rightarrow \lambda$ **do**
        $B_k^j \leftarrow Sig(D_i)[j-1]$
        if $B_k^j \notin \mathcal{L}$ **then**
            add B_k^j to $\mathcal{L}$
            create $V_{B_k^j}$ as an empty vector
        end if
        add $(H(id(D_i)), E_{K_{pub}}(rs_{ik})$ to $V_{B_k^j}$
    end for
end for
for all $B_k^j \in \mathcal{L}$ **do**
    $\pi_{B_k^j} \leftarrow HMAC_{K_{id}}(B_k^j)$
    add $(\pi_{B_k^j}, \mathcal{V}_{B_k^j})$ to secure index $\mathcal{I}$
end for
return $\mathcal{I}$

Finally, the data owner outsources the secure index $\mathcal{I}$ to the cloud server.

6.3.2 Secure Query Generation

This section explains the query generation process using trapdoors. As in the case of the index generation phase, we compute a signature for each document in the query. For sake of simplicity, we assume that the query is a

document as we search for k most similar documents to the document. The queried document is indeed represented as a set of terms that have different tf-idf scores.

Secure query generation is very similar to the index generation process. As in the flowchart given in Figure 6.2, after feature extraction MinHash functions are applied on the query terms which create a query signature. Let the set of query terms be F_q, then the query signature is calculated as

$$Sig(F_q) = \{h_{P_1}(F_q), \ldots, h_{P_\lambda}(F_q)\}.$$

Using query signature $Sig(F_q)$, the λ buckets corresponding to the query are determined. The bucket identifiers are then hidden with the HMAC function used in the index generation phase as

$$\pi_i = HMAC_{K_{id}}(B_i), \tag{6.5}$$

using the same key K_{id} of Equation (6.3).

The list of λ bucket identifiers is the secure query. Note that, the same set of λ MinHash functions are used both in index and query generations. Therefore, authenticated users need to know the permutations generated in index generation for MinHash functions. The set of permutations, which is referred as the *trapdoor*, is shared with all authenticated users, but hidden from the both non-colluding cloud servers.

In k-NN search, the tf-idf values of the terms in the query document are also used in the query generation phase. Recall that, output of a MinHash function, which determines the corresponding bucket identifier, is one of the terms in the query set. Therefore, tf-idf value of the output term is used in the query together with the bucket identifier. However, tf-idf values are also

sensitive information that may reveal the corresponding term. In order to hide the tf-idf values in secure queries, we apply an order preserving hashing (h_{op}) to each tf-idf value such that if $x > y$ then $h_{op}(x) > h_{op}(y)$. The order preserving function used in the protocol is defined as,

$$h_{op}(x) = MSB_{\zeta}(rx + r_2),$$

where r and r_2 are two random numbers ($r_2 < r$) and MSB_{ζ} is a function that returns the most significant ζ bits of the given input. While the same random r is used for all λ calls of h_{op} for a given query, r_2 is randomly chosen in each call of the function. If a deterministic function were used for hiding the tf-idf scores, for different MinHash functions that produce the same output, the corresponding scores would also be the same. This would eventually reveal that matching buckets contain data elements, which contain the same term. In order to avoid such leakage, we use a randomized order preserving function that outputs different, but close values for the same input with high probability.

The extracted ζ bit values are combined with HMACed bucket identifiers, resulting a vector of λ pairs as

$$V_q = \{(\pi_1, h_{op}(rs_{\pi_1}), \ldots, (\pi_{\lambda}, h_{op}(rs_{\pi_{\lambda}}))\}.$$

Algorithm 7 describes the process of query generation.

6.3.3 Secure Search

Search over the encrypted data is in fact performed over the secure index stored on the search server. As briefly mentioned previously, two non-colluding servers are employed in the search process, namely search server

Algorithm 7 Secure Query Generation

Require: F: feature set of keywords to be queried, h: λ MinHash functions, K_{id}: HMAC key
Ensure: V_q: Secure query

for all $D_q \in Q$ **do**
  Find $Sig(F)$ for D_q as in Algorithm 6
  Generate an odd random r
  for i = 1 to λ **do**
    $\pi_i \leftarrow HMAC_{K_{id}}(Sig(F)[i])$
    Generate random $r_2 < r$
    $rs_{\pi_i} \leftarrow$ score of $Sig(F)[i]$
    $h_{op}(rs_{\pi_i}) \leftarrow MSB_{\zeta}(rr_{\pi_i} + r_2)$
    set $V_q[i] \leftarrow (\pi_i, h_{op}(rs_{\pi_i}))$
  end for
end for
return V_q

and file server. While the search server stores the secure index generated by the data owner as explained in Section 6.3.1, the file server stores the actual encrypted data elements and knows the private key K_{priv} for the homomorphic encryption (i.e., Paillier cryptosystem) used to encrypt the relevancy scores. A user generates a secure query as explained in Section 6.3.2 and submits it to the search server. The search server works homomorphically on the secure index and calculates encrypted, unsorted relevancy scores for the list of the matching results. The encrypted results are then sent to the file server that possesses the private key of the homomorphic encryption. It decrypts and sorts the relevancy score list of the matching data items and sends those encrypted, top matching data items to the user.

Recall that, a secure query is of the form of bucket identifiers protected by HMAC function. The search server retrieves the buckets given in the query and homomorphically sums the encrypted scores of each document depending on their HMACed document identifiers. The query also contains relevancy

scores, which are first multiplied with the scores of data items in the secure index and then the homomorphic summation of the results is calculated.

For finding the scores of the same document in different buckets and summing their relevancy scores, the MapReduce computation framework is used. The search algorithm consists of two MapReduce phases. The first phase joins the query elements with the secure index elements. Multiplying randomized query scores with homomorphically encrypted index scores is also handled in this phase.

The second phase is responsible for homomorphically summing the scores of each document from different buckets. While the first phase consists of two mappers (query and index) and one reducer, the second phase consists of one mapper and one reducer.

In the first phase, the first mapper (i.e., query mapper) reads the given query vectors V_q, which contains the HMACed bucket identifiers and the corresponding relevancy scores as described in Section 6.3.2. Then, the mapper redirects each bucket id as a key and the corresponding relevancy score as a value prefixed with 'Q :' to the reducer as

$$< D_q, V_q > \longleftarrow < \pi_i, < Q : \pi_i, h_{op}(rs_{\pi_i}) >> .$$

The prefix 'Q :' is used to distinguish the pair from those that come from the secure index.

The second mapper (i.e., index mapper) reads each bucket id and their contents from the secure index. This mapper extracts the bucket id as a key and the HMACed data items' ids inside that bucket with their corresponding scores as value. Then, for each data item id in the bucket, a replication of the bucket id is created as a key and its corresponding encrypted score is included as value. This pair is then redirected to the reducer. This time the

values are prefixed with 'D :' to distinguish them from query bucket records as follows

$$< \pi_i, V_j > \longleftarrow < \pi_i, < D : H(id(D_j)), E_{K_{pub}}(rs_{ij}) >> .$$

The reducer of the first phase receives the elements within the same bucket (i.e., query elements and secure index elements), and adds them into two vectors such that documents prefixed with 'Q :' are mapped to the query vector V_Q, and documents prefixed with 'D :' are mapped to the data vector V_D. Then, the randomized scores in the query vector (i.e., $h_{op}(rs_{\pi_i})$) are homomorphically multiplied with the scores of the data items in the data vector. The output of the first phase combines the data item id pairs (i.e., query element id and index element id) as key, and the relevancy scores (e.g., encrypted multiplication results) as value. The output of this phase is in the following format:

$$<< Q, H(id(D_j)) >, h_{op}(rs_{\pi_i}) \times E_{K_{pub}}(rs_{ij}) > .$$

In the second phase, the aggregated encrypted score is calculated by summing the resulting encrypted relevancy scores of the same pairs. In order to calculate the sum, the mapper only redirects all the key value pairs that come from the previous phase as they are, without any modification. After the shuffling process, the reducer receives the records that have the same key (i.e., with the same data item identifier), and the encrypted relevancy scores for each bucket. Utilizing homomorphic properties of Paillier encryption, encrypted aggregated relevancy score (rs_{Agg}) of each data item is calculated.

The final output of the reducer is in the following format

$$<< Q, H(id(D_j)) >, E_{K_{pub}}(rs_{Agg_j}) > .$$

This list of pairs with encrypted scores are then sent to the file server, which decrypts and sorts the accumulated relevancy scores. Then the final results with the highest relevancy scores are sent to the user.

6.4 Security Analysis

In the proposed method, all the actual data items that are stored in the file server are encrypted with a semantically secure symmetric encryption scheme (e.g., AES with CBC and CTR modes) prior to outsourcing. Therefore, an adversary can learn no information using the encrypted data elements. However, meta-data called the secure index, that is used for applying secure search operation, is also outsourced together with the encrypted data. The secure index is a number of buckets where each bucket stores a list of pairs $H(id(D_i)), E_{K_{pub}}(rs_i)$ that are mapped to that bucket. Here, $H(id(D_i))$ is a hashed pseudo identifier for a data element D_i and $E_{K_{pub}}(rs_i)$ is the corresponding relevancy score which is encrypted with the Paillier encryption scheme. Since the relevancy scores are also encrypted with a semantically secure encryption method, the encrypted values are indistinguishable from random numbers and hence do not reveal any information. The only information that an adversary may use for an attack is the pseudo identifiers of the data items.

During the secure index generation process, each data item is mapped to exactly λ different buckets. Therefore, it is not possible to make any statistical analysis using the number of occurrences of a pseudo identifier. On

the other hand, each bucket corresponds to a single feature/term. Therefore, the number of data item identifiers in each bucket is correlated with the frequency of the corresponding feature in the whole data set. With high probability, buckets that correspond to a feature that occurs in larger number of data elements have larger number of identifiers. However, each feature may correspond to several (at most λ but can be fewer) different buckets and it is hard to distinguish feature - bucket identifier correspondence. Moreover, the secure index and the actual encrypted data elements are stored in two different and non-colluding servers. The search server, which is the adversary for our security model, can only access the secure index and therefore cannot learn the mapping between the actual encrypted data items and their pseudo identifiers. The proposed method provides data confidentiality as defined in Section 6.2 since the adversary cannot learn the features of the data items using the secure index.

In order to enhance the security, a possible patch can be applied to also hide the actual number of data identifiers in each bucket. This can be done by adding a number of dummy pseudo identifiers to buckets with fewer number of elements to perturb the cardinality statistics of the buckets. The relevance scores of the dummy identifiers should be set to random encryptions of zero, hence they will be automatically omitted in the top list of relevancy in the file server side after being sorted based on their relevancy scores. Another important point is that each of the dummy identifiers must be assigned to exactly λ different buckets to make them indistinguishable from genuine identifiers. Note that, inclusion of extra dummy elements will increase the computation cost of search operation as aggregated score should also be calculated for dummy elements using homomorphic addition of encrypted relevancy scores. Also the communication between search server and file server will increase as

the encrypted scores of dummy identifiers will also be sent. Another possible solution to hide this leakage is to include all data element identifiers in the data set to all the buckets. The corresponding relevance scores will be set to the encryption of zero if the data item does not originally map to that bucket. However, this will drastically increase both the computation and communication costs between the two servers, and hence being impractical.

A search query, independent of the query type, is a list of λ bucket identifiers. The information leaked to the search server from a query is that the query contains the features that correspond to the buckets given in the query. However, as explained for data confidentiality, the bucket identifiers and the corresponding list of data item pseudo identifiers do not reveal the actual features. Therefore, the proposed scheme provides query confidentiality.

The queries are generated in a deterministic manner hence different queries that search for the same set of features will be the same. This leaks the so called *search pattern* such that the frequency of the buckets accessed is leaked to the search server. A query randomization method that hides the search pattern is proposed and analyzed in [45]. The same approach can also bu used in this setting with the cost of increasing the storage requirements as the approach requires utilizing higher number of buckets.

6.5 Experiments

In order to assess the performance of the proposed scheme, we build a small cluster of 3 nodes and Cloudera CDH4 for cluster management. The cluster have two nodes with Xeon processor E5-1650 3.5 GHz with 12 cores 15.6 GB of RAM and 1 TB hard disk and one node with Core i7 3.07 GHz with 8 cores, 15.7 GB of RAM and 0.5 TB hard disk. On each node, Ubuntu

12.04 LTS, 64 -bit operating system is installed and Java 1.6 JVM is used.

In our experiments, we used the Enron data set [34] for evaluating the text based operations of the proposed method. This data set contains a collection of $517,000$ real e-mail files. The size of each file changes from 4 KB to 2 MB. Although the actual data set size is about 200 GB, the size of the secure index is very small since only the information of bucket identifiers and corresponding data element identifier and score pairs are stored.

Initially, the entire data set is processed to determine the features of each data item in what is known as the feature extraction phase. First, the data set is cleaned from the mail headers and stop words. The terms are stemmed to their roots using Snowball stemmer, which is a string processing programming language included in the Lucene [35] library. The *tf-idf* values of each term in each e-mail file are then calculated and stored on sparse vectors. After the feature extraction phase, secure index is generated as explained in Section 6.3.

For the accuracy experiments we consider two metrics namely *precision* and *recall*. Intuitively, precision is the ratio of correctly found matches over the total number of returned matches and recall is the ratio of correctly found matches over the total number of expected results (i.e., the ground truth). Both precision and recall are real numbers between 0 and 1, where a higher value means better accuracy.

We assume that the server receives several search requests at any time which is in line with the big data context. Therefore instead of processing each query individually, we group 200 individual queries and process a bulk search for all. Bulk search operations have a great positive effect on the throughput of the system as the distributed file systems and parallel programing models do not benefit relatively small number of such operations.

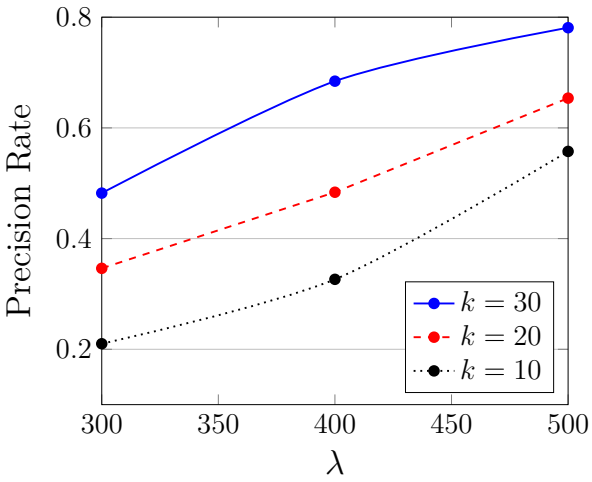


Figure 6.3: Average precision rates for k-NN search with different λ and k

As the number of terms in queries is significantly many (as a document in a query can contain many words with relatively high tf-idf scores), a larger value of λ needs to be used in order to satisfy an equivalent level of accuracy. Furthermore, both precision and recall metrics are equivalent as the number of returned matches and the number of actual matches are both equal to k. We therefore used a single metric, namely precision, to evaluate the accuracy of the k-NN search method. The accuracy of k-NN search in our experiments is illustrated in Figure 6.3 for different values of k and λ.

Figure 6.3 demonstrates that increase in λ has a positive effect on accuracy as expected due to the properties of locality sensitive hash functions. Similarly, increase in k has also a positive effect on the accuracy. The proposed method can find the most similar data items (i.e., nearest neighbors) to the queried item, but possibly in a different order with respect to the actual results. Therefore, for small values of k, some of the actual most similar matches may not fall in the top k slots and leads to a negative effect on the accuracy. However, for relatively larger values of k, most of the top matching data items are usually accommodated in the top k slot and thus leads to high

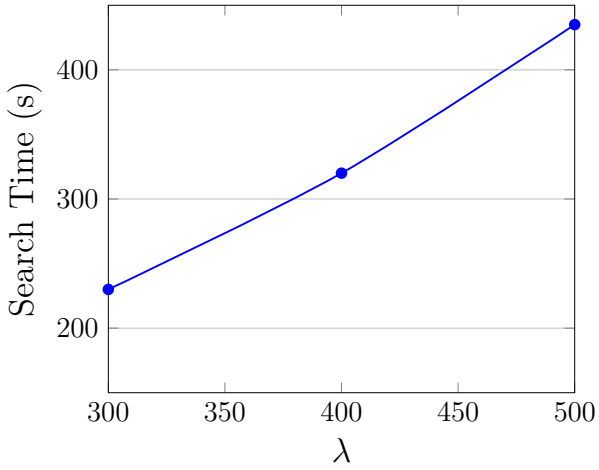


Figure 6.4: Average search time for kNN search with different λ

accuracy rates.

The execution times of k-NN search for different values of λ and k are illustrated in Figure 6.4.

6.6 Conclusion

In this work, we addressed the problem of applying an existing privacy-preserving technique for secure document similarity search operation on encrypted cloud data. We utilized the secure similarity search method based MinHash functions. We developed an implementation of the proposed secure search algorithm for the Hadoop distributed file system and the MapReduce programming model. We tested the implementation in a three-node Hadoop cluster with the Enron email data set and demonstrate the effectiveness and scalability of the system.

Although the timing results are high the proposed method is still promising in the sense that the method can scale to larger data sets with more documents provided that more computing nodes in the cluster are available.

In addition, as more computation power at a reasonable cost will be available in the future, more secure solutions as proposed in this chapter will be feasible.

Chapter 7

EFFICIENT, SECURE DOCUMENT SIMILARITY SEARCH UTILIZING Z-ORDER SPACE FILLING CURVES

In this chapter, we propose another secure document similarity search method in a scenario where a database of documents is hosted by a semi-trusted cloud server. Users submit their queries consisting of documents to the cloud, which performs a search operation over the database index and returns the most similar documents to those in the queries. As queries, database and database index are protected by a symmetric encryption technique, we provide data, query and index confidentiality. The method is amenable to parallel computing technologies where database is kept in a distributed file system. Thus, it is scalable to large databases, which makes it a good candidate for big data

applications. The experimental results show that proposed method introduces no significant overhead to execution times. However, the effectiveness of the method is deteriorated due to the trade off between security and accuracy. To remedy this, we propose several techniques to improve the accuracy by allowing local computation in the user side and modifying database index construction.

7.1 Introduction

Recently, cloud computing became a popular technology for relatively inexpensive, robust and reliable storage and computation. Providing efficient and fast computing power at low cost, it compels parties, who are unwilling to invest their limited budget in hardware, to outsource their data and applications to cloud services. However, the cloud service providers are usually considered as semi-trusted entities that may try to learn from the stored data.

Processing and/or analyzing data over relational databases is an important problem that has extensively been considered in the literature for a long time. However, the security implications had been disregarded, to a large extent, until recently as data outsourcing was not a common practice. Therefore, with the increasing use of and reliance on cloud, it is now a necessity to provide techniques for privacy-preserving processing of data, prior to outsourcing. Similarity search is one of the foremost problems need to be addressed when private and/or sensitive data are outsourced to cloud.

More specifically, contents and features (e.g. words) of documents, which are outsourced, can be private and need to be protected from the cloud service (*the cloud* henceforth), which can be done by encryption. On the other hand,

conventional encryption usually hinders any operation on documents, including similarity search, in which documents with similar features are searched. Therefore, secure and searchable metadata (e.g. index) for the documents are constructed, which do not leak information about the document contents and features by itself.

An important requirement of secure search operations is that it has to scale to massive data/document sets to be expedient in big data applications, which are the current focus of interest in industry as well as academia.

This chapter considers the problem of determining the most similar documents in an encrypted data set given a query document, without revealing the features (i.e. terms, words) of neither the data set elements nor the query document. Search for similar documents is implemented using a signature called *ZOLIP key* which is computed using the document features and their importance for the document. The essence of the ZOLIP key is that similar documents, having the same important features, have significant portions of the key in common. Prior to outsourcing, a ZOLIP key per data set entry is generated and encrypted by the data owner. Then the search is applied over encrypted ZOLIP keys without using the encrypted data itself.

The proposed method does not protect search pattern privacy, as ZOLIP keys are generated deterministically. Therefore, features of same importance levels can be linked across queries leading to information leak, which can be exploited given background information and known statistical properties of the data set. Guaranteeing search pattern privacy is generally considered as a difficult problem, which can be detrimental for scalability of the solution. Therefore, we only provide a simpler, but scalable solution for big data applications.

The work here can be considered as an extension of our work in [33], which

provides no security, but focuses on fast and scalable document similarity search solutions. Based on the same theoretical foundation, here we provide a simple technique to secure documents as well queries.

7.2 Problem Definition

In this section we formulate the problem and present the definitions related to our proposal for document similarity search operation used throughout the chapter. The notation used in the chapter is listed in Table 7.1.

Table 7.1: Common Notations

$\mathcal{D}$	data set.
Q	query set.
d_i	i^{th} document in the data set.
q_i	i^{th} query document in the query set.
Z_i	ZOLIP key for document d_i
λ	number of Z-order iterations.
δ	number of terms in the data set.
v_i	random vectors of δ elements.
ζ	size of signature groups derived from ZOLIP key

Note that we adopt the terminology and definitions introduced in [33], which also proposes a document similarity search method, but without any security properties.

Definition 16 (Secure Similarity Search) *Let $\mathcal{D}$ be the outsourced data set and $d_1, \ldots, d_n$ be the n records in D and $Q =< q_1, \ldots, q_m >$ be a query with m documents. Further let $Sig_e(d_i)$ be an encryption for document having the signature Z_i. Also, $Sig_e(\mathcal{D})$ and $Sig_e(Q)$ stand for set of signatures for the data set and the query, respectively. Then, the secure similarity search*

(SSS) protocol is defined as:

$$SSS(Sig_e(\mathcal{D}), Sig_e(Q)) \rightarrow < d_{11}, \ldots, d_{1k}, \ldots, d_{m1}, \ldots, d_{mk} >,$$

where d_{ij} is the identifier of the j^{th} nearest document in $\mathcal{D}$ to $q_i \in Q$.

Definition 17 (Data Confidentiality) *Given a k-similar document protocol, let $\mathcal{D}$ be the data set outsourced to the cloud. An SSS protocol provides data confidentiality if the contents of the documents $d_i \in D$ are not revealed to the cloud server.*

Definition 18 (Query Confidentiality) *Given a k-similar document protocol, let Q be the set of documents in a query. An SSS protocol provides query confidentiality if the contents of the documents $q_i \in Q$ are not revealed to the cloud server.*

7.3 Secure ZOLIP Similarity Search

In this section we present the secure version of the ZOLIP algorithm introduced in [33]. Secure similarity search is performed using secure search index generated by the owner of the data, which is the set of encrypted document signatures. Data set is also encrypted and uploaded to the cloud along with the secure search index. This way, the cloud owner is prevented from accessing document and query contents, while it is still possible to perform search for similar documents.

7.3.1 Secure Index and Query Generation

The main idea of our work is to create signatures to represent document in the data set, which are used to compare documents and determine their sim-

ilarity. We use ZOLIP keys as document signatures, introduced by Alewiwi, et al. [33] which are, simply speaking, just bit vectors. The proposed method consists of two main phases, feature extraction and secure ZOLIP key construction as shown on the left hand side of Figure 7.1.

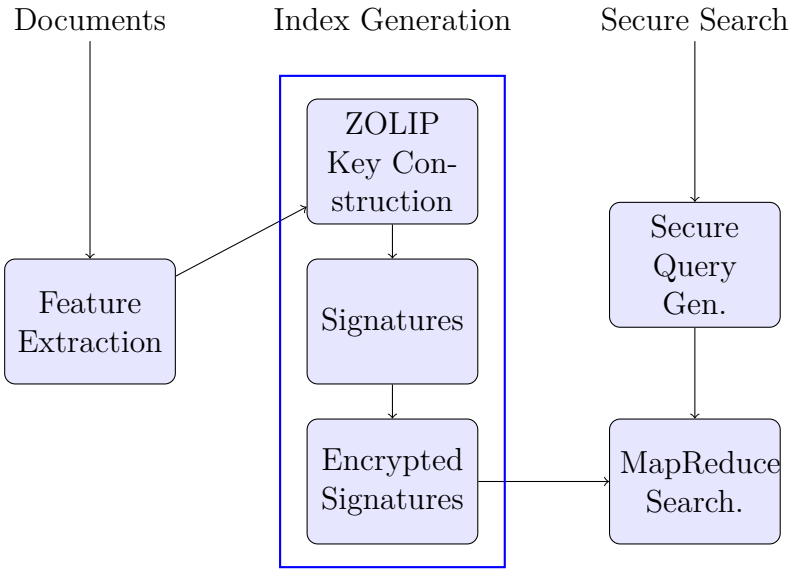


Figure 7.1: Flowchart of secure index and query generation

In document similarity, features are words in documents and the feature extraction phase processes documents and determines the words and their tf-idf values. As a result, we obtain a sparse vector for each document, in which non-zero values are tf-idf scores for the corresponding words in the document. In the second phase, the ZOLIP key is first constructed by bit-wise interleaving the tf-idf values of important words for the corresponding document and the resulting bit sequence is stored in a sparse vector.

Let $Z_i = \{b_1 \dots b_{\delta\lambda}\}$ be the ZOLIP key for document d_i, where δ is the number of terms in the dataset and λ is the number of Z-Order iterations. We partition the ZOLIP key into groups or chunks, each of which contains

ζ consecutive bits. Then, the ZOLIP key can be written as

$$Z_i = \{Z_1 \dots Z_{\lceil \frac{\delta \cdot \lambda}{\zeta} \rceil}\},$$

where Z_i is a chunk of ζ consecutive bits.

In order to prevent signatures from revealing any information about the document, its content, the words they contain and their tf-idf scores we use a deterministic encryption scheme on signature chunks, Z_i. For fast computation and short outputs, we use an HMAC function, which takes a secret key K_i and the signature chunk Z_i for $i = 1 \dots \lceil \frac{\delta \cdot \lambda}{\zeta} \rceil$, and returns a ciphertext block. Consequently, the secure signature of document d_i is given as

$$Sig_e(d_i) = \{\text{HMAC}(K_1, Z_{i,1}), \dots, \text{HMAC}(K_{\lceil \frac{\delta \cdot \lambda}{\zeta} \rceil}, Z_{i,\lceil \frac{\delta \cdot \lambda}{\zeta} \rceil})\}$$

We use a different HMAC keys for each chunk and a chunk with all-0 bits are ignored. Note also that document id's are also encrypted. Finally, the encrypted documents and their encrypted ids and signatures sent to the cloud.

In the secure query generation operation (see Figure 7.1), the same steps are applied to obtain encrypted signatures for the documents in the query. In the next section we describe the document similarity search in more detail.

7.3.2 Secure Search

The secure search operation is composed of three map reduce phases. The first phase tries to find duplicate and/or near duplicate documents (i.e. two documents with exactly the same important terms should have the same ZOLIP key), and it contains two mappers and one reducer.

The first mapper reads and parses the secure index (i.e. the set of en-

crypted signatures $Sig_e(d_i)$ for the entire data set), then it passes the following key-value pair

$$< E(d_i), Sig_e(d_i) > \rightarrow < Sig_e(d_i), D : E(d_i) >$$

to the reducer. Here, $Sig_e(d_i)$ serves as the key and $D : E(d_i)$ as the value, where $E(d_i)$ is the encrypted document id and the prefix "D" is used to distinguish the index from the query.

The second mapper reads the secure signatures in the query in a similar manner. Its output has the same format, except for the fact that the value part is prefixed with "Q"

$$< E(d_i), Sig_e(q_i) > \rightarrow < Sig_e(d_i), Q : E(q_i) > .$$

The reducer of the first phase receives documents sharing the same encrypted signature. It groups the documents prefixed with "Q" and documents prefixed with "D" into two separate sets. For each document in the "Q" set, it tries to find k similar documents from the "D" set; an operation which may not be always successful.

If the reducer finds sufficient number of similar documents (i.e. k similar or near duplicate documents), it returns the encrypted document id as the key and, as the value, the original encrypted document signature concatenated with the list of documents having the same signature as the query document. The suffix "Y" is added to the value part indicating that the search is successful and the desired number of similar documents are found. The main aim of attaching "Y" to the result is to avoid the next phase, which is executed only if the first phase falls short of finding k similar documents.

In the first phase is not successful, the search result contains the prefix

"N" to signal that we need to move to the next phase. Consequently, the reducer output of the first phase has the following form

$$< E(d_i), < Sig_e(d_i), \{E(d_j)\}, (Y/N) >> .$$

The second phase is also composed of two mappers and one reducer. The main aim of this phase is to find the documents having signature chunks in common. The first mapper reads the original encrypted signatures, while the second mapper reads the output of the first phase. The first mapper parses chunks of signature values (i.e. HMAC(K_l, Z_l) for $l = 1, 2, \ldots, \lceil \frac{\delta \cdot \lambda}{\zeta} \rceil$), and then sends chunks as key and encrypted document ids as values to the reducer. The resulting mapper output has the following form

$$< E : (d_i), HMAC(K_l, Z_{l,1}) > \rightarrow$$
$$< HMAC(K_l, Z_{l,1}), D : E(d_i) > .$$

The second mapper in this phase reads the output of the first phase and checks if there are documents not having k similar documents. It parses the signatures $Sig_e(d_i)$ of the encrypted documents with insufficient number of similar documents and obtains signature chunks $HMAC(K_l, Z_{i,l})$ for $l = 1, 2, \ldots, \lceil \frac{\delta \cdot \lambda}{\zeta} \rceil$. It then forms the pairs

$$< HMAC(K_l, Z_{i,l}), Q : E(d_i) >$$

and sends them to the reducer.

Documents having the same encrypted signature chunks indicate that they share some parts of the ZOLIP key and thus possibly sharing some important terms. Generally, higher number of common signature parts point

out that the documents are very similar. Upon receiving the outputs of the mappers, the reducer creates two lists; the *query list* for encrypted signature chunks from the query documents and the *data set list* for those of documents in the dataset. For each matching chunks from the query list and the data set list, the reducer forms the following pair

$$<< E(d_i), E(d_j) >, 1 >,$$

where the encrypted ids of documents containing the same encrypted chunk are taken as the key and the integer "1" as the value of the pair.

The last phase is responsible for joining the results of the first and second phases. Again, this phase has two mappers and one reducer. The first mapper reads the first phase output, parses the record values with "Y" and extracts encrypted document ids. It then concatenates the encrypted query document id with all the encrypted document ids as a key. The value of this mapper is another literal value "$\lceil \delta \cdot \lambda/\zeta \rceil$" (i.e. identical ZOLIP keys as all signature chunks match). The result of this mapper will have the form

$$<< E(d_i), E(d_j) >, \lceil \delta \cdot \lambda/\zeta \rceil > .$$

The second mapper reads the output of the second phase and redirects it without any modification to the reducer. In the last step, the reducer of the last phase reads the pairs of the encrypted document ids. Then for each pair, the algorithm finds the sum of the value parts from the previous mappers and outputs them in the form

$$<< E(d_i), E(d_j) >, sum > .$$

The user is sent the output of the last phase sorted depending on the sum parts of the pair. Here, the value *sum* for $< E(d_i), E(d_j) >$ gives the number of common signature chunks in two documents.

7.4 Security Analysis

The actual documents stored in the cloud are assumed to be encrypted with a semantically secure symmetric encryption scheme (e.g., AES) prior to outsourcing. Therefore, the cloud and any adversary that captures them do not learn much information from the ciphertexts as the secret key is not available to them. However the secure index, composed of encrypted signatures and needed for secure search operation, is also outsourced together with the encrypted documents. The encrypted signatures consist of HMACed ZOLIP key chunks for documents. The signatures do not give any information about the terms in the corresponding ZOLIP keys as long as the universal dictionary is shuffled using a random permutation.

Documents having common signature chunks indicate that they possibly share terms. As the ZOLIP key is grouped as chunks before encryption there can be false positives, namely documents having common signature may not share the corresponding terms with certain probability, which depends on the chunk size. The larger chunk size increases this probability while decreasing the accuracy due to the increasing number of false positives. False positives serve as randomizing effect on the secure index as well as on the queries. This makes the adversary's job harder as it is difficult to know whether a document contains a term or not for certain.

In summary, the proposed method provides data confidentiality, since the adversary can learn neither the terms a document contains nor their scores.

However, the secure index and queries can give partial information about the documents such as an idea about the approximate number of important terms it contains. An adversary, therefore, can differentiate between documents with many important terms and those with fewer number of important terms.

The queries are generated in a deterministic manner, hence queries containing the same documents can easily be identified. Hence, so called search pattern is leaked. The proposed method provides only fundamental security properties, namely data, index and query confidentiality.

7.5 Experimental Results

In order to assess the performance of the proposed scheme, we build a small cluster of 3 nodes and Cloudera CDH4 for cluster management. The cluster has two nodes with Xeon processor E5-1650 3.5 GHz with 12 cores 15.6 GB of RAM and 1 TB hard disk and one node with Core i7 3.07 GHz with 8 cores, 15.7 GB of RAM and 0.5 TB hard disk. On each node, Ubuntu 12.04 LTS, 64 -bit operating system is installed and Java 1.6 JVM is used.

In our experiments, we used the Enron data set [34] for evaluating the proposed method. This data set contains a collection of $517,000$ real e-mail files. The size of each file changes from 4 KB to 2 MB. Although the actual data set size is about 200 GB, the size of the secure index is very small as only the signatures for ZOLIP key chunks with non-zero values are stored.

We selected Z-order iteration level $\lambda = 8$ and similarity level $k = 10$. We performed the experiments using unencrypted signature chunks as only bit string comparisons are needed independent of whether we use encrypted or unencrypted chunks. As the comparison of bit strings constitutes only a small portion of the overall execution time and no encryption operation is

performed in the server side, we did not actually use the HMACed ZOLIP key. Also, we report only the experimental results for the search operation in the server side excluding the operations such as feature extraction and query construction.

Initially, the data is processed in what referred as the feature extraction phase (see Figure 7.1). In feature extraction, first the data is cleaned from the mail headers and stop words. The terms are stemmed to their roots using the snowball stemmer included in the Lucene [35] library. The *tf-idf* values of each term in each file are then calculated and stored on sparse vectors. After the feature extraction phase, secure index and queries are generated as explained in Section 7.3.1.

The performance and accuracy of the algorithms are measured for different values of ζ in our experiments. First, we measured the performance of the algorithm by measuring the execution time using different ζ values to detect the effect of increasing ζ over the performance. In our experiments we used $\zeta = \{2, 3, 4\}$ and employed queries of 1,000 documents assuming that a typical cloud server receives multiple search requests from many users. Figure 7.2 shows the effect of changing the ζ value on the overall execution time of the search operation in the server side. The figure shows that larger ζ values slightly improve the time performance as the number of comparisons of signature chunks is decreased. When compared with the timing results in [33], which provides a document similarity search algorithm without any security, the execution times are comparable for similar experimental settings. On the other hand, the communication complexity is negatively affected as the query sizes increase due to encryption of the ZOLIP key.

We also depicted the accuracy of the proposed in Figure 7.3, which shows that larger values of ζ deteriorate the accuracy. This result is expected as

the increasing sizes of signature chunks lead to increase in false positives. Also, the larger chunk sizes will result in documents for which we cannot find sufficient number of similar documents.

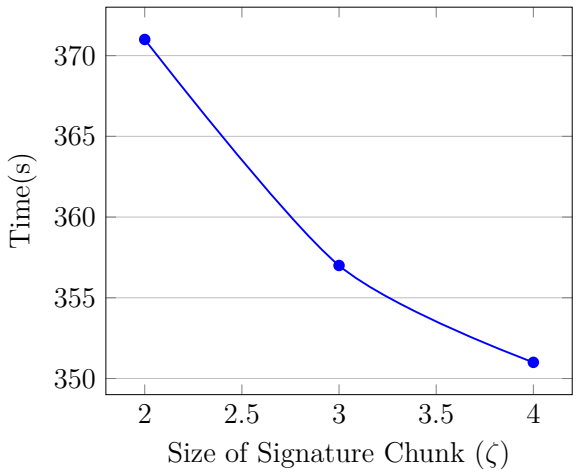


Figure 7.2: Time Complexity

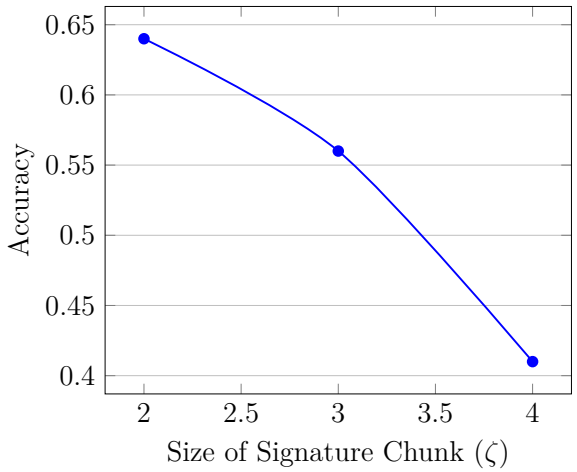


Figure 7.3: Average Accuracy Rate

As seen from the figures, large values of ζ have a significantly negative effect on accuracy and slight improvement in time performance. Also, from security point of view large values of ζ has certain advantages as they increase false positive rates leading to less partial information leak. Therefore, we

can conclude that there is a trade off between accuracy and security. A decision for picking the correct ζ depends on the application requirements. Nevertheless, the experimental results clearly show that only small values of ζ leads to reasonable accuracy levels.

To increase accuracy, we propose to use larger values of k than specified in the query and return all documents found after the search to the user. Then, local computation at user side will become a necessity as the number of returned documents exceeds the specified k. Also, the user needs to remove the false positives, namely dissimilar documents. Some client side computation is acceptable and sometimes desirable from security point of view as long as the computation requirements are not prohibitively high.

7.6 Conclusion

We present the secure version of a previously proposed document similarity search method by Alewiwi et al [33]. The method is almost as fast and efficient as the original method in [33], while there is a trade off between the security and accuracy. While we can improve the accuracy of the method by adjusting the chunk size of the document signatures, the original version will always be better as we introduce additional false positives due to encryption. As a remedy, the accuracy can further be improved if the algorithm returns more documents than requested to the user that, in turn performs additional processing over a limited number of documents. This is acceptable if local computation is feasible at the user side. Lastly, the method also increases the query and index sizes, due to the encryption of small chunks of document signatures individually. On the other hand, both query and index sizes remain at acceptable levels as document signatures are sparse bit vectors.

Chapter 8

CONCLUSION AND FUTURE WORK

In this thesis, we addressed the secure document similarity problem in the scenario that document set is outsourced to a honest but curious cloud server. We targeted applications involving processing of big data, for which any solution must be efficient and scalable. Therefore, all proposed solutions feature algorithms that are amenable to parallel computation, hence scalable.

We used the MapReduce parallel programming model due to its simplicity for our parallel implementation of the proposed algorithms. As a parallel computation platform, we utilized Hadoop framework thanks to its accessibility and relative ease of use and maintenance. Using MapReduce and Hadoop, we demonstrated that our algorithms can scale to large sizes of data sets. We proposed four different solutions for document similarity search: while the first one provides no security, the other three solutions provide protection, however, with different security properties. The last solution, in fact, is a secure version of the first one.

We first proposed an efficient filtering technique for document similarity

search in Chapter 4. The solution does not provide any security and focuses on efficiency and scalability of the similarity search operation. As we used cosine similarity metric to compare documents based on the terms and their tf-idf values, we obtained good accuracy results. The proposed filtering technique prevents the computation of cosine similarity of two dissimilar documents, and focusing only on potentially similar documents. The experimental results obtained using two known data sets demonstrate that the method is promising for big data applications. Although it has limitations for documents with no important terms, future research can remedy this by using hybrid solutions.

For *secure* document similarity search, we developed three algorithms. The first solution, covered in Chapter 5, considers two different security levels. The basic method uses plain sketches and the other method offers an enhanced security protection by using encrypted sketches, where each sketch element is encrypted with the Paillier encryption method. The first algorithm providing a basic level security is a feasible solution in many applications, while the other more secure algorithm shows good theoretical accuracy. Unfortunately, our experimental results demonstrate that the more secure version is not practical due to its heavy computational requirements.

In Chapter 6, we adapted an existing privacy-preserving multi-keyword search technique for secure document similarity search problem on outsourced data. More specifically, we utilized MinHash functions, which are used in the literature [45] [46] for secure multi-keyword search operations in similar settings. The proposed algorithms offer better security properties if two non-colluding servers implementing cloud operations are available. The experimental results are promising as far as execution times and accuracy are concerned. In addition, they also show that it is possible to reconcile effi-

ciency and effectiveness of the search method and advanced security features, at least for moderately large data sets.

The last method given in Chapter 7, is in fact, a secure version of the algorithm introduced in Chapter 4. To preserve efficiency and scalability of the original algorithm, the new algorithm offers only basic protection of document, index, and query privacy, allowing search and access patterns to leak. The initial experimental results demonstrate that the secure algorithm has acceptable computational complexity (at least comparable to that of the original scheme). Nevertheless, it is also clear that the accuracy of the method should be improved. We suggest a solution to improve the accuracy, but the matter calls for much deeper treatment, which we recommend for future research.

As final remarks, secure document search operation is a challenging research area that features interesting problems, especially in the cloud setting of outsourced data. It is a difficult problem as there is usually a trade off between either security and computational efficiency or security and accuracy. While advanced encryption algorithms can improve accuracy in addition to security, they are usually not amenable to fast implementations for big data applications. In this thesis, we proposed scalable solutions, which provide promising results for data sets of moderate size. Reconciling accuracy and security with computational efficiency for even larger data sets is the subject of future research.

Bibliography

[1] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using mapreduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, (New York, NY, USA), pp. 495–506, ACM, 2010.

[2] T. Jung, X. Mao, X. Li, S. Tang, W. Gong, and L. Zhang, "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation," in *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*, pp. 2634–2642, 2013.

[3] Y. Yang, H. Li, W. Liu, H. Yao, and M. Wen, "Secure dynamic searchable symmetric encryption with constant document update cost," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 775–780, Dec 2014.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[5] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pp. 918–929, VLDB Endowment, 2006.

[6] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, (New York, NY, USA), pp. 743–754, ACM, 2004.

[7] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, (Washington, DC, USA), pp. 5–, IEEE Computer Society, 2006.

[8] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 131–140, ACM, 2007.

[9] F. Angiulli and C. Pizzuti, "An approximate algorithm for top-k closest pairs join query in large high dimensional data," *Data and Knowledge Engineering*, vol. 53, no. 3, pp. 263–281, 2005.

[10] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 4, pp. 599–608, 2010.

[11] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, (New York, NY, USA), pp. 131–140, ACM, 2008.

[12] S. Zhu, J. Wu, H. Xiong, and G. Xia, "Scaling up top-k cosine similarity search," *Data and Knowledge Engineering*, vol. 70, no. 1, pp. 60–83, 2011.

[13] T. Elsayed, J. Lin, and D. W. Oard, "Pairwise document similarity in large collections with mapreduce," in *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, (Stroudsburg, PA, USA), pp. 265–268, Association for Computational Linguistics, 2008.

[14] B. Yang, J. Myung, S.-g. Lee, and D. Lee, "A mapreduce-based filtering algorithm for vector similarity join," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '13, (New York, NY, USA), pp. 71:1–71:5, ACM, 2013.

[15] R. Li, L. Ju, Z. Peng, Z. Yu, and C. Wang, "Batch text similarity search with mapreduce.," in *APWeb* (X. Du, W. Fan, J. W. 0001, Z. Peng, and M. A. Sharaf, eds.), vol. 6612 of *Lecture Notes in Computer Science*, pp. 412–423, Springer, 2011.

[16] R. Baraglia, G. De Francisci Morales, and C. Lucchese, "Document similarity self-join with mapreduce," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 731–736, Dec 2010.

[17] T.-C. Phan, L. d'Orazio, and P. Rigaux, "Toward intersection filter-based optimization for joins in mapreduce," in *Cloud-I'13*, pp. 2–2, 2013.

[18] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security*, pp. 442–455, Springer, 2005.

[19] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in

Proceedings of the 13th ACM conference on Computer and communications security, pp. 79–88, ACM, 2006.

[20] W. Jiang, M. Murugesan, C. Clifton, and L. Si, "Similar document detection with limited information disclosure," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pp. 735–743, IEEE, 2008.

[21] M. Murugesan, W. Jiang, C. Clifton, L. Si, and J. Vaidya, "Efficient privacy-preserving similar document detection," *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, vol. 19, no. 4, pp. 457–475, 2010.

[22] S. Buyrukbilen and S. Bakiras, "Secure similar document detection with simhash," in *Secure Data Management*, Lecture Notes in Computer Science, pp. 61–75, 2014.

[23] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pp. 139–152, 2009.

[24] F. Li and X. Xiao, "Secure nearest neighbor revisited," *2014 IEEE 30th International Conference on Data Engineering*, vol. 0, pp. 733–744, 2013.

[25] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," *CoRR*, vol. abs/1307.4824, 2013.

[26] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[27] C. Zhang, F. Li, and J. Jestes, "Efficient parallel knn joins for large data in mapreduce," in *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, (New York, NY, USA), pp. 38–49, ACM, 2012.

[28] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Efficient and accurate nearest neighbor and closest pair search in high-dimensional space," *ACM Trans. Database Syst.*, vol. 35, pp. 20:1–20:46, July 2010.

[29] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge: Cambridge University Press, 2012.

[30] R. A. Brown, "Hadoop at home: Large-scale computing at a small college," *SIGCSE Bull.*, vol. 41, pp. 106–110, Mar. 2009.

[31] "Apache Hadoop." http://hadoop.apache.org.

[32] M. Bellare, R. Canetti, and H. Krawczyk, "Message authentication using hash functions- the hmac construction," *CryptoBytes*, vol. 2, 1996.

[33] M. Alewiwi, C. Orencik, and E. Savas, "Efficient top-k similarity document search utilizing distributed file systems and cosine similarity," *Cluster Computing*, pp. 1–18, 2015.

[34] "Enron Dataset." http://www.cs.cmu.edu/∼./enron/.

[35] "Lucene." http://lucene.apache.org/.

[36] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, Dec. 2004.

[37] F. Falchi, R. Perego, C. Lucchese, F. Rabitti, and S. Orlando, "A metric cache for similarity search," in *In LSDS-IR*, 2008.

[38] C. Örencik, M. Alewiwi, and E. Savas, "Secure sketch search for document similarity," in *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*, pp. 1102–1107, IEEE, 2015.

[39] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[40] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *ADVANCES IN CRYPTOLOGY - EUROCRYPT 1999*, pp. 223–238, Springer-Verlag, 1999.

[41] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, vol. 21, no. 3, pp. 333–358, 2012.

[42] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, (New York, NY, USA), pp. 139–148, ACM, 2008.

[43] M. Franz, P. Williams, B. Carbunar, S. Katzenbeisser, A. Peter, R. Sion, and M. Sotakova, "Oblivious outsourced storage with delegation," in *Financial Cryptography and Data Security* (G. Danezis, ed.), vol. 7035 of *Lecture Notes in Computer Science*, pp. 127–140, Springer Berlin Heidelberg, 2012.

[44] M. Kuzu, M. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pp. 1156–1167, April 2012.

[45] C. Orencik, A. Selcuk, E. Savas, and M. Kantarcioglu, "Multi-keyword search over encrypted data with scoring and search pattern obfuscation," *International Journal of Information Security*, 2015.

[46] C. Orencik and E. Savas, "An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking," *Distributed and Parallel Databases*, vol. 32, no. 1, pp. 119–160, 2014.